Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE VALLADOLID

DESARROLLO E INTEGRACIÓN DE FUNCIONES EXTERNAS PARA ACELERAR TIEMPOS DE EJECUCIÓN DE MODELOS VENSIM

Trabajo de Fin de Grado

Grado en Ingeniería Informática
Mención Ingeniería de Software

Alumno:

Arturo Manteola Llamas

Tutor:

César Llamas Bello

2021

Resumen

La simulación de sistemas dinámicos es una disciplina dentro de la ingeniería que trata de conseguir y hacer funcionar modelos que permiten predecir el comportamiento tanto de sistemas por construir como sistemas existentes, pero también permite profundizar en cómo comprendemos sistemas que ya existen consiguiendo mayores volúmenes de datos a analizar de los que podemos obtener de la realidad. Sus áreas de aplicación incluyen sistemas físicos, químicos y modelos económico sociales del mundo real entre muchos otros. En este último contexto, el proyecto LOCOMOTION trata de proporcionar modelos y simulaciones que permitan ayudar a tomar decisiones a los agentes políticos, y también a concienciar al público de las consecuencias del cambio climático entre otros.

Uno de los grandes problemas de la simulación de estos sistemas es la gran cantidad de tiempo y recursos que se pueden llegar a requerir para obtener simulaciones de modelos complejos con grandes cantidades de datos de entrada. En este proyecto se aborda la construcción y el diseño de una biblioteca de funciones para el software que se emplea en el proyecto LOCOMOTION, que es Vensim. La finalidad es reducir los cuellos de botella de potencia de cálculo mediante una extensión DLL de Vensim que incluya las funciones con mayor potencial de mejora y que permita su ampliación en el futuro.

Al evaluar las prestaciones de esta biblioteca sencilla se ha podido comprobar que el uso que se hace de OpenBlas mediante la DLL dentro de Vensim permite reducir el tiempo de cómputo de la operación que consume más tiempo hasta el 50.12 % del tiempo en el caso de prueba más grande, esperando mejoras mayores en modelos con mayores volúmenes de datos.

Abstract

Dynamic system simulation is a discipline within the engineering field that focuses on making models that allow predicting the behaviour of both systems to be built or existing systems, but they also serve the purpose of making us learn about systems by letting us gather larger volumes of data to analyze than what we can get from observing reality. It can be applied on physical, chemical or socioeconomic models among many others. The LOCOMOTION project delves into socioeconomic models with the purpose of providing models and simulations that help political agents make decisions, and raising public awareness about climate change among others.

One of the biggest problems in the simulation of these systems is the great amount of time and resources that may be needed to get simulations from complex models with huge amounts of input data. This project focuses on the design and developement of a library of functions for the software used in the LOCOMOTION project, Vensim. Its purpose is to reduce bottlenecs caused by a lack of computing power by tackling the functions deemed to have the greatest margin of improvement and that allows future expansions to be made.

When evaluating the performance of this simple library it has been possible to verify that the use given to OpenBlas within the DLL used by Vensim reduces computation time by up to 50,12 % of the time in the largest test case, with the expectations that having bigger input data volumes means greater reductions in time.

Presentación

La simulación de sistemas dinámicos es una disciplina dentro de la ingeniería que trata de conseguir y hacer funcionar modelos que permiten predecir el comportamiento tanto de sistemas por construir como sistemas existentes, pero también permite profundizar en cómo comprendemos sistemas que ya existen consiguiendo mayores volúmenes de datos a analizar de los que podemos obtener de la realidad. Sus áreas de aplicación incluyen sistemas físicos, químicos y modelos económico sociales del mundo real entre muchos otros. En este último contexto, el proyecto LOCOMOTION trata de proporcionar modelos y simulaciones que permitan ayudar a tomar decisiones a los agentes políticos, y también a concienciar al público de las consecuencias del cambio climático entre otros.

El Grupo de Energía, Economía y Dinámica de Sistemas (GEEDS) de la Universidad de Valladolid está formado por un grupo de profesores e investigadores. Participan en un programa de investigación europeo, LOCOMOTION, que trata entre otras muchas cosas de completar los modelos desarrollados en otro proyecto de investigación europeo MEDEAS.

Uno de los grandes problemas de la simulación de estos sistemas es la gran cantidad de tiempo y recursos que se pueden llegar a requerir para obtener simulaciones de modelos complejos con grandes cantidades de datos de entrada. En este proyecto se aborda la construcción y el diseño de una biblioteca de funciones para el software que se emplea en el proyecto LOCOMOTION, que es Vensim. La finalidad es reducir los cuellos de botella de potencia de cálculo mediante una extensión DLL de Vensim que incluya las funciones con mayor potencial de mejora y que permita su ampliación en el futuro.

Al evaluar las prestaciones de esta biblioteca sencilla se ha podido comprobar que el uso que se hace de OpenBlas mediante la DLL dentro de vensim permite reducir el tiempo de cómputo de la operación que representa el mayor cuello de botella, inversión matricial, hasta el 50.12 % del tiempo en el caso de prueba más grande, esperando mejoras mayores en modelos con mayores volúmenes de datos.

Estructura de la memoria

En este documento se recogen tanto los resultados de investigar acerca de bibliotecas de álgebra lineal, estudiar con detalle el funcionamiento de los DLLs de Vensim para poder utilizar funciones externas y acerca de los entornos de programación disponibles para la creación de DLLs en Windows como la planificación y el desarrollo de un DLL para Vensim con varias funciones.

En el primer capítulo se pone en contexto lo que es Vensim, los usos que tiene, sus puntos fuertes y sus problemas. También se habla de los objetivos del trabajo, la planificación de este y se entra en algo de profundidad sobre lo que se espera conseguir en cada iteración del DLL como el tiempo que se espera tardar.

El segundo capítulo está dedicado a los resultados de comparar las bibliotecas de álgebra lineal disponibles para averiguar cuál es la más adecuada para el proyecto según ciertos criterios preestablecidos. Una vez elegida la biblioteca se habla de particularidades que tiene y que hay que tener en cuenta.

El tercer capítulo está dedicado a analizar mucho más en profundidad Vensim para que se pueda entender qué aspecto tiene, cómo se modela en el y, sobre todo, la estructura y el funcionamiento del código de ejemplo para la creación de un DLL. También se habla de otros programas utilizados para el desarrollo del DLL, de los modelos que serán utilizados como casos de prueba y del desarrollo del primer prototipo.

El cuarto capítulo explica el resto de detalles del software utilizado, las medidas iniciales de rendimiento obtenidas para la inversión matricial y del desarrollo del segundo prototipo.

En el quinto capítulo se habla de las funciones que se desean realizar durante el trabajo de forma más concreta, y se explica al detalle el funcionamiento y el desarrollo de la función de inversión matricial, junto a las pruebas utilizadas y un análisis de la mejora de rendimiento que aporta la función. Además, se habla de los otros dos prototipos, de todos los avances realizados en ellos y los problemas que han impedido completarlos en el tiempo límite para el trabajo.

En el sexto capítulo se realiza una retrospección del proyecto y se habla de las posibles mejoras que se pueden realizar en trabajos futuros.

A continuación se encuentran los anexos que explican cómo instalar y utilizar el software que se ha creado, mencionado y utilizado durante este trabajo. Y, por último, se dispone de una bibliografía con las referencias más útiles usadas durante el proyecto.

Agradecimientos

Lo primero agradecer a mi familia todo el apoyo dado durante la carrera, asegurándose de que siempre tenga recursos suficientes para poder avanzar sin problemas, y a mi novia, quien ha estado siempre animándome y apoyándome en los momentos malos.

También agradecer a todos los amigos que he conocido en la carrera, motivándome a seguir adelante, pudiendo mantener conversaciones interesantes tanto de informática como de cualquier otra cosa y con quienes he vivido muchos momentos que no olvidaré.

Quiero agradecer la oportunidad que ha sido participar en este interesante proyecto al grupo GEEDS, quienes han aportado mucho al avance de este interesante proyecto y se han ofrecido a apoyarlo con una beca.

Quiero agradecer a mi tutor de este Trabajo de Fin de Grado, César Llamas, toda la ayuda ofrecida durante el desarrollo del proyecto. Siempre ha estado disponible para resolver todas las dudas que me han ido surgiendo y hemos tenido unas reuniones muy productivas para hablar del rumbo que este trabajo estaba tomando. También agradecer a Jesús Vegas su participación en muchas de estas reuniones y sus contribuciones. Muchas gracias a ambos.

Y por último agradecer al resto de profesores de la escuela de ingeniería informática quienes se han esforzado en enseñarnos conceptos muy importantes para nuestro futuro sin los cuales este trabajo hubiera resultado imposible.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Objetivos, contexto y propuesta

En este capítulo se va a explicar cuáles son los objetivos de este proyecto, el problema que se intenta solucionar y la planificación del trabajo que se ha realizado.

Para ello, se dará algo de contexto relacionado con el problema primero.

Luego se listarán los prototipos que se planearon hacer, mencionando de cada uno la funcionalidad del prototipo, el objetivo que se logra al finalizar el prototipo, la estimación de tiempo para realizarlo y el tiempo invertido en desarrollarlo.

1.1. Problemas de la simulación de sistemas

La simulación de sistemas dinámicos trata de modelar un sistema dinámico cuyas variables cambian con el tiempo y resolver el valor que tomarán estas variables tras el paso del tiempo. El modelo que dicta cómo interactúan las variables entre sí y cómo son modificadas a través del tiempo es independiente de el estado inicial de las variables.

En la práctica se usa para cosas muy diversas como pueden ser predecir el estado en un futuro cercano de un sistema real y poder actuar con antelación, para animaciones, videojuegos, averiguar el resultado que se obtendría de un hipotético caso inicial o para insertar estados iniciales con cierto nivel de aleatoriedad. Este último caso permitiría encontrar patrones o relaciones que no son aparentes con los datos que se puedan llegar a extraer de la realidad.

Todos estos casos pueden llegar a exigir cierto nivel de optimización, bien sea por la necesidad de que la simulación se adelante a los acontecimientos de la realidad o tenga que llevar cierto ritmo o porque se realicen grandes cantidades de simulaciones con grandes cantidades de datos iniciales diferentes.

1.2. Optimización de cálculos en Vensim

Vensim [2] es un programa orientado al modelado de sistemas dinámicos cuya primera versión fue lanzada en 1990. Si bien es cierto que todos estos años de actualizaciones han culminado en un programa con muchas funciones muy útiles a la hora de desarrollar modelos, también hay que tener en cuenta que

este programa no fue pensado para el cómputo de volúmenes de datos de tamaños tan grandes como los que podemos obtener hoy en día.

Por estas razones las funciones encargadas de operaciones matemáticas complejas no están optimizadas, mucho menos para grandes volúmenes de datos.

Además, Vensim incluye funciones con funcionalidades muy diversas pero que implican recorrer matrices varias veces de forma muy dispersa, lo cual constituye un gran problema en el caso de que esta matriz sea tan grande que no pueda estar cargada entera en la memoria del programa.

1.3. Propuesta: Ampliar mediante DLLs

Por suerte, Vensim ofrece la posibilidad de importar al programa funciones creadas por el usuario por medio de un DLL, que tiene que seguir cierta estructura y cumplir ciertas condiciones para funcionar adecuadamente con el programa. [7]

Además, Vensim ofrece una plantilla de ejemplo con diversas funciones básicas. Esta plantilla está autodocumentada y es a su vez un manual acerca de cómo crear estas funciones externas definidas por el usuario.

Teniendo en cuenta que los cuellos de botella se encuentran en operaciones complejas con matrices muy grandes, la propuesta realizada para mejorar el rendimiento de Vensim es encontrar una biblioteca de funciones ya creada que nos ofrezca operaciones matriciales ya optimizadas y usarla para la creación de un DLL que ofrezca alternativas a las funciones de Vensim pero con un rendimiento mayor, especialmente en los casos de matrices de gran tamaño.

1.4. Objetivos

Antes de realizar la planificación del trabajo, se propusieron una serie de objetivos que se quiere cumplir con el proyecto. El objetivo principal es la propuesta realizada por GIR GEEDS consistente en crear un DLL para acelerar el cálculo de inversión matricial en Vensim.

El resto de objetivos han sido resultado de pensar en los requisitos necesarios para llegar a crear el DLL, en los pasos a seguir o en las posibles ampliaciones de este en caso de haber tiempo para realizarlas.

Los objetivos propuestos han sido los siguientes:

- Realizar un estudio de las librerías disponibles que permitan realizar ese tipo de cálculos.
- Realizar un estudio de la tecnología disponible para la creación de DLLs.
- Crear un DLL que permita añadir funciones externas a Vensim.
- Desarrollar una función externa que permita realizar la operación de inversión matricial en menor tiempo que Vensim para matrices grandes.

- Desarrollar una función externa que permita colapsar dimensiones de una matriz en menor tiempo que Vensim para matrices grandes.
- Redactar un manual de instalación para la librería de cálculo utilizada.
- Redactar un manual de uso de las funciones externas creadas.
- Crear una serie de funciones dentro del DLL que permitan almacenar matrices dentro del espacio de memoria del DLL.

1.5. Planificación del trabajo

En cuanto a la cantidad de tiempo disponible para realizar el proyecto se decidió dedicar 15 horas semanales durante un tiempo de entre tres y cuatro meses.

Debido a que se plantearon metas que conseguir basadas en las metas anteriores, fue fácil ver que el modelo de planificación más apropiado para el proyecto es uno basado en prototipos, ya que se puede definir un prototipo para cada una de estas metas y estas metas tienen una naturaleza iterativa.

Además, esto permite estimar cuánto tiempo va a llevar realizar el proyecto sumando el tiempo esperado necesario para completar cada uno de estos prototipos.

A continuación se explicarán con poco detalle los prototipos relacionados con cada meta. Al principio del proyecto se definieron 6 prototipos, aunque durante el desarrollo la mayor parte de lo que era el cuarto prototipo (aquí nombrado «Tercer prototipo (II)») se desarrolló durante el tercer prototipo de manera que la finalización del tercer prototipo resultó ser casi simultánea a la finalización del siguiente prototipo planeado.

Es por eso que el orden de los prototipos sigue el orden de los 5 desarrollados durante el proyecto pero se ha dividido el tercer prototipo en dos para reflejar la planificación.

También se mencionará la estimación del tiempo necesario para completar cada uno de los prototipos junto a el tiempo que llevó completarlos. Los prototipos serán explicados con mucho más detalle en capítulos posteriores.

1.5.1. Primer prototipo: Conexión con Vensim

El primer prototipo que se desea obtener se detalla en el Capítulo 3 y en el se pretende simplemente compilar y conectar la plantilla que nos ofrece Vensim al propio Vensim, ya que la plantilla va a ser usada como base para el desarrollo.

Completar este prototipo significa que se ha encontrado una herramienta de desarrollo y compilación adecuada y que todos los bugs que surgen de que la plantilla no está completamente actualizada han sido solucionados.

Dentro del desarrollo de este prototipo se incluye el tiempo estimado y utilizado para investigar acerca de las herramientas de desarrollo de DLLs y el tiempo necesario para configurarlas adecuadamente.

Tiempo estimado: Dos semanas.

Tiempo utilizado: Tres semanas.

1.5.2. Segundo prototipo: Integración de OpenBlas

El segundo prototipo plantea la creación de un programa muy simple que muestre en una terminal el resultado de una operación realizada a través de OpenBlas.

Completar este prototipo significa que se ha logrado instalar OpenBlas de una forma compatible con el entorno de desarrollo que se ha usado, y que ahora se dispone de un pequeño entorno de pruebas donde poder realizar operaciones sencillas en OpenBlas para comprobar sus resultados de forma directa e inmediata

Dentro del desarrollo de este prototipo se incluye el tiempo usado para la investigación acerca de la librería a usar, además de la instalación de esta. Se menciona el uso de OpenBlas en este documento, sin embargo a la hora de planificar aún no se había decidido la librería concreta a utilizar, por lo que el nombre de este prototipo era «Integración de la librería».

Tiempo estimado: Tres semanas.

Tiempo utilizado: Tres semanas. .

1.5.3. Tercer prototipo: Integración de OpenBlas en Vensim

El tercer prototipo se podría considerar una combinación de los dos anteriores, y el objetivo es crear un DLL usado por Vensim en el que se ejecute una llamada a una función de OpenBlas.

Completar este prototipo significa que es posible integrar bibliotecas externas en Vensim a través de un DLL.

Tiempo estimado: Una semana.

Tiempo utilizado: Cuatro semanas.

1.5.4. Tercer prototipo (II): Desarrollo de la función de inversión

Este prototipo será el primer prototipo que pueda ser usado en el proyecto, y el objetivo es crear un DLL que pueda realizar la operación de inversión matricial.

Completar este prototipo significa que ya tenemos un artefacto directamente útil, desde el que se pueden realizar pruebas para comprobar el rendimiento de Vensim con y sin DLL y, en caso de que se logre alguna mejora, integrarlo en un proyecto real.

Tiempo estimado: Dos semanas.

Tiempo utilizado: Dos días (Al realizar el anterior prototipo, se realizó gran parte de este).

1.5.5. Desarrollo de los manuales

Tras tener un DLL que ya puede ser incorporado al proyecto, lo siguiente es redactar un manual de uso y de instalación. Una vez realizado, se planeó una reunión con un par de integrantes del equipo para darles el manual de instalación y observar cómo interactúan con el.

Mientras lo instalan, tomar nota de los puntos donde han tenido problemas, las dudas que han tenido y las sugerencias que realicen al final de la reunión. El manual será redactado en inglés para su uso de forma internacional.

Tiempo estimado: Una semana.

Tiempo utilizado: Dos semanas.

1.5.6. Cuarto prototipo: Almacenamiento de matrices en la DLL

El cuarto prototipo propone crear una forma alternativa de llamada a la función, mediante la cual se pueden ahorrar movimientos de matrices entre el DLL y Vensim.

Completar este prototipo supone la base para una forma alternativa de uso del DLL que se podrá ampliar en un futuro, y que puede aportar una mejora de rendimiento notable en ciertas situaciones.

Tiempo estimado: Dos semanas.

Tiempo utilizado: Cuatro semanas.

1.5.7. Quinto prototipo: Desarrollo de la función SUM

El quinto prototipo tiene como objetivo crear una versión de una función ya existente en Vensim, que pese a ser más restrictiva en cuanto al número de cosas que puede realizar, sea más eficiente que Vensim en los casos en los que sea adecuado usarla.

Completar este prototipo supone tener otra mejora aplicable directamente al proyecto.

Tiempo estimado: Dos semanas.

Tiempo utilizado: Una semana.

1.5.8. Presupuesto

Se ha utilizado para el desarrollo de este trabajo una máquina virtual de dos cores y 8 gb de ram, con Windows 10. Se ha encontrado que una máquina virtual de estas características cuesta 26€ al mes, y habiendo sido utilizada durante 4 meses nos deja con un coste de 104€ a lo largo del proyecto.

Una licencia de Vensim DSS de por vida cuesta 1995€. Esta licencia ha sido cedida por el grupo GEEDS para la realización del proyecto. Para calcular los costes se ha calculado el coste mensual de la

licencia con una amortización de 10 años, es decir, 199,5€ al año o 16,63€ redondeando al mes. Dado que la licencia ha sido cedida por un total de 4 meses, se calcula el coste de la licencia como 66,52.

El sueldo medio anual de un ingeniero informático recién graduado en 2019 ha sido en torno a 21000€. Dividido por las 1950h anuales de trabajo que se realizan se obtiene un coste efectivo de 10,80€ cada hora. Multiplicado por las 255 horas que se han dedicado al proyecto obtenemos un coste de mano de obra de 2750€.

Sumando estos tres costes obtenemos un coste total para el proyecto de 2924,52€

1.6. Riesgos

Los riesgos que se han planteado son:

- OpenBlas sea incompatible con Vensim.
- No se pueda lograr una mejora de rendimiento a través de un DLL externo.
- El uso de un DLL externo altere de alguna manera el funcionamiento de Vensim, modificando parámetros que no debería aunque sea de forma mínima.
- La instalación de la librería es demasiado difícil para ser instalada individualmente solo con ayuda del manual de instalación.

Capítulo 2

Bibliotecas de álgebra matricial. OpenBlas

En este capítulo se muestran diversas bibliotecas que se han considerado para su uso. Se comparan con respecto a las necesidades de nuestro proyecto y se detalla la elegida.

Con respecto a la biblioteca que se va a utilizar se consideran los siguientes parámetros de elección prioritarios desde el punto de vista del desarrollo:

- Libre disposición, sin limitación de licencias de uso.
- Disponible para diversas arquitecturas.
- Nivel de madurez aceptable.
- Comunidad de soporte.
- Que tenga continuidad.

En cuanto a sus capacidades de cómputo se priorizarán las bibliotecas con respecto a estos factores:

- Velocidad de ejecución.
- Consumo de memoria.

2.1. Bibliotecas disponibles

La biblioteca necesaria para este proyecto es una que pueda realizar operaciones de álgebra lineal sobre matrices. Existe una biblioteca llamada BLAS (Basic Linear Algebra Subprograms) [3] que especifica un conjunto de operaciones con sus implementaciones, y sus rutinas, que son estándares y de bajo nivel, se usan en muchos casos como base para crear bibliotecas de operaciones de álgebra lineal.

Existen bastantes bibliotecas que cumplen este estándar BLAS [6], estas son las más relevantes dentro de las que se han tenido en cuenta:

- **ATLAS:** ATLAS es una biblioteca que implementa las funciones de BLAS de una forma optimizada y es la primera biblioteca cuya instalación genera código optimizado para la máquina en la que está siendo instalada. Sin embargo, la última versión estable fue publicada en 2016.

- **LAPACK:** LAPACK [4] forma otro estándar de funciones de álgebra lineal, sin embargo, LAPACK surgió en 1992 como una actualización a una biblioteca anterior, LINPACK, que estaba basada en BLAS.

  A día de hoy se sigue actualizando y se usa como otro estándar disponible.

- **MKL:** Intel oneAPI Math Kernel Library [5] es una biblioteca desarrollada inicialmente por Intel, aunque ahora pertenece a opeAPI. Esta biblioteca no tiene una instalación muy compleja y tiene muy buen rendimiento en procesadores Intel, sin embargo ofrece tiempos de cómputo mucho peores en procesadores AMD.

  Como la biblioteca no será instalada en procesadores Intel exclusivamente, se optó por otra opción.

- **OpenBlas:** OpenBLAS surgió de una rama de la biblioteca GotoBLAS, la cual es una implementación de BLAS cuyo desarrollo cesó en 2010. Sin embargo, OpenBLAS sigue actualizándose actualmente por el Institute of Software, Chinese Academy of Sciences (ISCAS).

  OpenBLAS, al igual que ATLAS, ofrece una instalación que optimiza la biblioteca al hardware en el que está siendo utilizado, y ofrece tanto una implementación de BLAS como de LAPACK.

  Además, ofrece un rendimiento similar a MKL en las funciones pertenecientes a BLAS en procesadores Intel, aunque en las funciones pertenecientes a LAPACK ofrezca un rendimiento peor. [1]

  En procesadores AMD, ofrece un rendimiento mejor.

2.2. Decisión de OpenBLAS

Las razones por las que se ha usado OpenBlas son:

- OpenBlas es de libre disposición, utiliza una licencia de tipo BSD de 3 cláusulas, que permite el uso y la redistribución de este software siempre y cuando se incluya el texto asociado a la licencia.

- Los desarrolladores pretenden lograr que el software funcione lo mejor posible en la mayor cantidad de hardware posible.

- Al instalarse se optimiza para cada máquina, incluyendo el uso de instrucciones específicas para ciertos tipos de hardware.

- Su instalación además deja la biblioteca lista para ser paralelizada.

- Los desarrolladores trabajan activamente en el proyecto y tiene una comunidad activa que propone mejoras y reporta los bugs, a veces hasta con la solución propuesta.

- Está basado en bibliotecas con muchos años de historia que han demostrado que sus algoritmos funcionan correctamente y son muy eficientes.

- OpenBLAS está actualmente siendo desarrollado y actualizado, por lo que también se optimiza para hardware actual y, previsiblemente, será actualizado para hardware que vaya saliendo al mercado.

- Ofrece un rendimiento similar a Intel oneAPI MKL en procesadores Intel, siendo mucho mejor en AMD.

2.3. Formas de instalación y diferencias entre ellas

OpenBlas ofrece varias formas de instalación, entre las cuales cada una ofrece distintas ventajas, inconvenientes y propiedades. [7]

Para su uso con Visual Studio era necesario que la instalación generase un DLL. Visual Studio no permite usar OpenBlas de otra manera, ya que no permite el uso de instrucciones optimizadas para solo ciertos tipos de procesadores, así que rechaza el uso del código sin compilar.

Teniendo en cuenta esto, se descartaron todas las que no generasen un DLL en su instalación, y se priorizó que la instalación generase código optimizado para la máquina.

Visual Studio ofrece su propia forma de instalar OpenBlas, pero no solo no generaba código optimizado ya que el mismo no lo admite, sino que además instalaba una versión parcial de OpenBlas con la que no parecía funcionar la rama de LAPACK de OpenBlas, que era la que interesaba para la función de inversión matricial.

Además, esta opción ni está mencionada en la página de los desarrolladores de OpenBlas ni en ningún manual, por lo que los desarrolladores no ofrecen de forma explícita ninguna garantía de que funcione correctamente.

De entre las formas que sí están incluidas en el manual que ofrecen los desarrolladores de OpenBlas, tenemos dos opciones que generan un DLL: Instalando por Miniconda3, que parecía ser la opción principal, o instalando con MinGW [17] que permite usar el compilador GCC en Windows, que en el propio manual explica que es un método de instalación experimental que no ha sido testeado minuciosamente.

Sin embargo, OpenBlas fue instalado de ambas maneras con el objetivo de comparar ambas versiones, tanto en dificultad y tiempo de instalación como en rendimiento en Vensim.

No se logró hacer que ninguna de las dos versiones funcionara en Visual Studio y hubo que cambiar de entorno de desarrollo a CodeBlocks. Se profundiza en los problemas específicos y en las razones que motivaron a este cambio en el Capítulo 3

Debido a ser la única forma de instalación que generaba un DLL que funcionase correctamente en un entorno que permitiese generar un DLL para Vensim, la forma de instalación elegida para su uso en el proyecto es a través de Miniconda3.

2.4. Uso de OpenBlas en C

Como ya se mencionó en este capítulo, BLAS y LAPACK se usan como estándares de bibliotecas de álgebra lineal, y OpenBlas es una biblioteca que cumple con ambos estándares, aunque ofrece instalar solo el código pertinente a uno de los estándares.

Por eso, para su uso en C hay que llamar a las funciones siguiendo la nomenclatura establecida por estos estándares, predecido de uno de los siguientes prefijos:

- `cblas_` si quieres usar una función de BLAS codificado en C
- `blas_` si quieres usar una función de BLAS codificado en Fortran
- `lapack_` si quieres usar una función perteneciente a LAPACK
- `lapacke_` si quieres usar una función perteneciente a LAPACKE

La diferencia entre LAPACK y LAPACKE es que LAPACKE ofrece más facilidades para ciertas operaciones, sobre todo si las matrices no se almacenan por orden de columnas. LAPACKE permite indicar en cada operación el tipo de orden que siguen las matrices.

Capítulo 3

Plataforma y creación de DLLs para la plataforma Vensim

En este capítulo se explica cómo funciona Vensim, qué problemas tiene a la hora de ser usado para el proyecto LOCOMOTION y cómo se ha planteado desarrollar la solución.

Además, se analizará el código de ejemplo dado por Vensim para ser usado como base para crear un DLL que expanda la funcionalidad de Vensim a partir del cual se crea el primer prototipo explicado también en este Capítulo.

También se explicará en detalle los casos de prueba usados como referencia.

3.1. Esquema general de Vensim

Vensim es un software de simulación de sistemas dinámicos. Ofrece a sus usuarios una interfaz gráfica para modelar los sistemas donde se crean diagramas causales con nodos representando variables y flechas indicando las interacciones y relaciones entre ellos.

La Figura 3.1 muestra el resultado de abrir uno de los casos de prueba que se explicará más adelante. Se puede observar los nodos y las flechas que los relacionan, y que cada variable puede utilizar y ser utilizada por varios otros nodos o variables.

Dentro de estos nodos se define de qué forma depende esta variable de las anteriores o cuál es su valor en caso de que sea un nodo de origen. Estos nodos pueden realizar operaciones que comprenden desde leer datos de una tabla externa hasta juntar datos en una matriz o dividirlos en varias, pudiendo realizar todo tipo de operaciones en el proceso e incluso tener valores que dependan de la cantidad de pasos que se han simulado.

En la Figura 3.2 se ve el resultado de utilizar la herramienta «Ecuations» sobre uno de los nodos. Esta es la forma principal de programar un modelo en Vensim, ya que es en esta ventana donde se indica el comportamiento que tiene este nodo.

Este nodo en concreto está programado para realizar una inversión matricial usando el DLL.

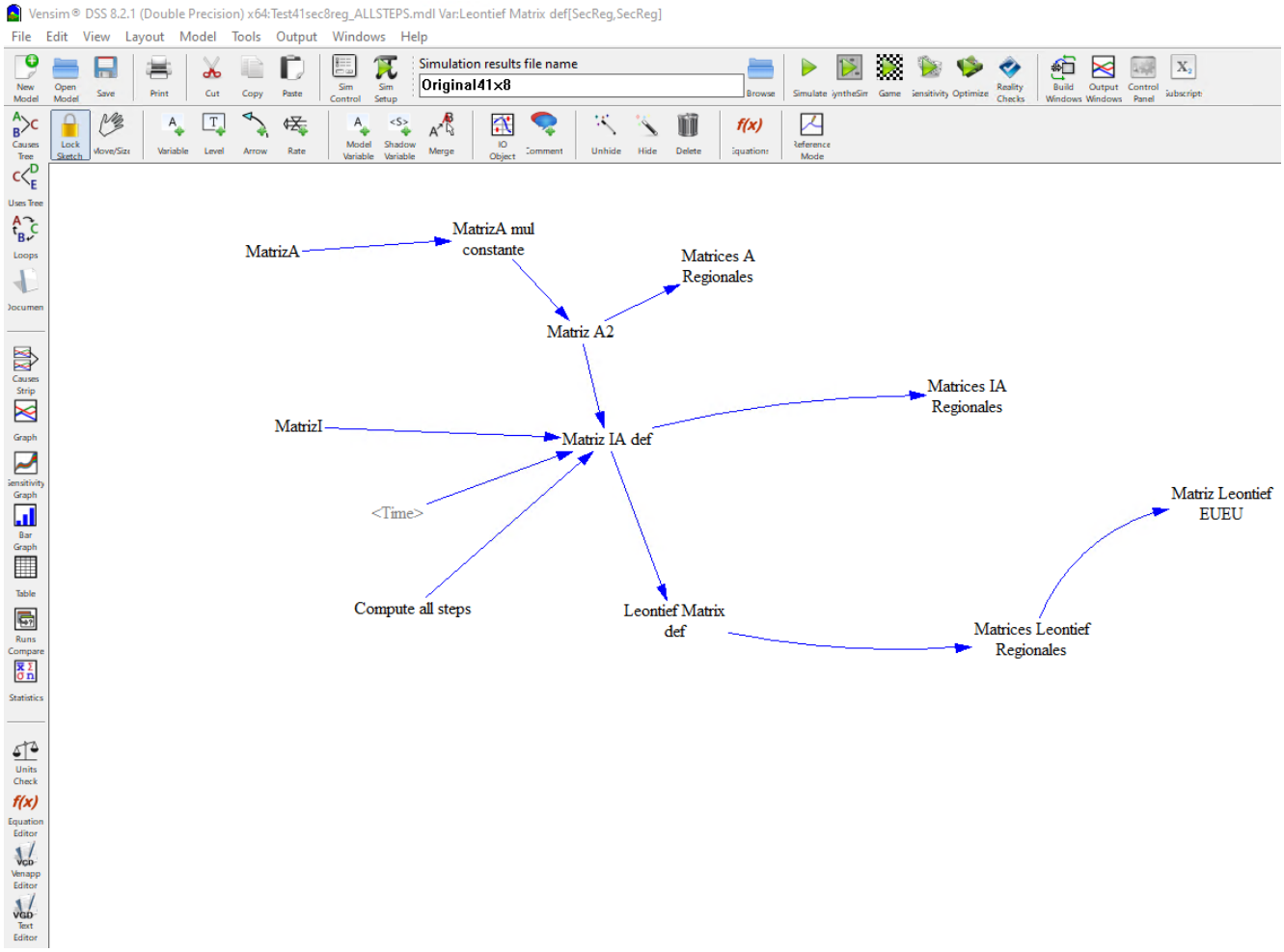


Figura 3.1: Interfaz gráfica de Vensim con una red de variables lista para su simulación en plataforma Windows.

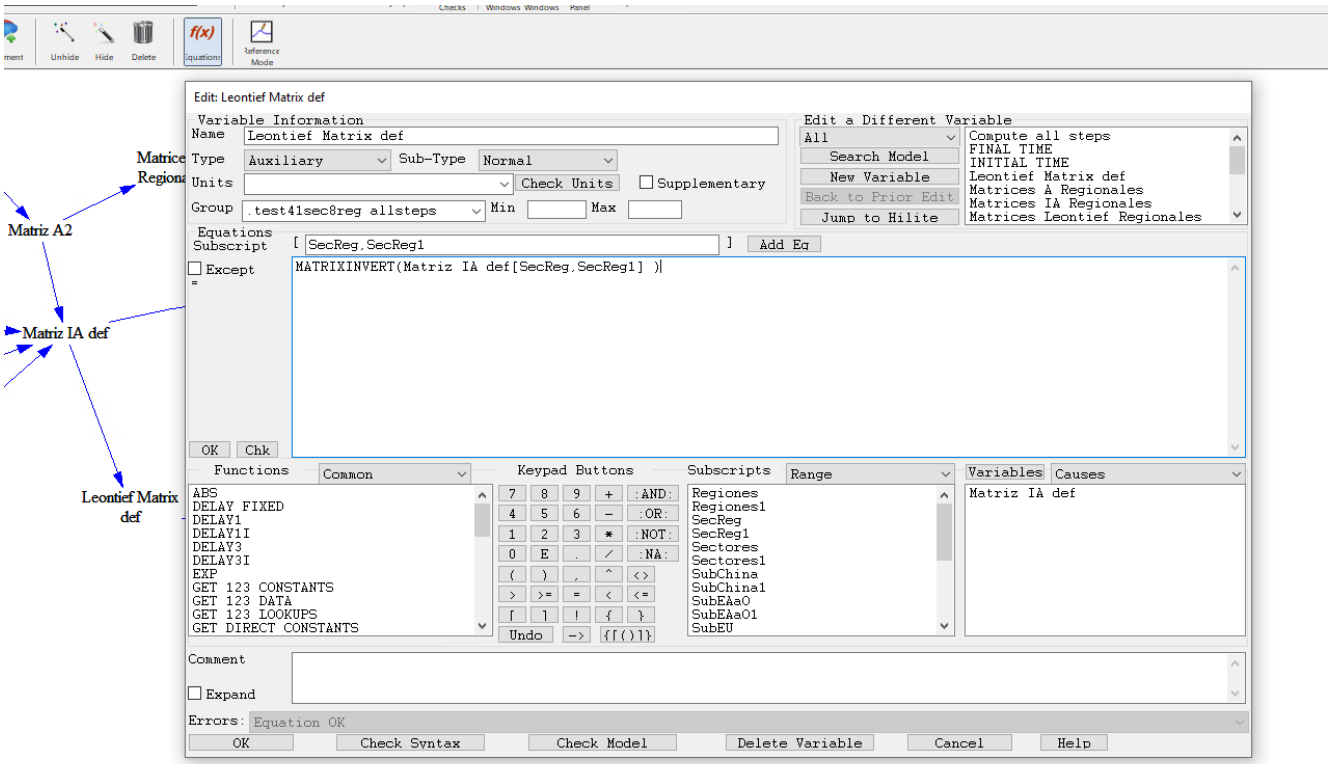


Figura 3.2: Nodo expandido (`Leontief Matrix def`) correspondiente al diagrama de la Figura 3.1.

Muchas de estas opciones, pese a ser muy útiles para un desarrollador experto en Vensim, no afectan de ninguna forma al funcionamiento del DLL ni a la forma de programarlo. Otros solo son para facilitar la creación del modelo, pero no aportan ninguna funcionalidad, solo elementos para autocompletar o para asistir al usuario. Por todo esto solo se van a explicar los apartados relevantes al proyecto de esta ventana.

En el apartado `Name` se puede ver y modificar el nombre del nodo que se ha abierto. En este caso el nodo abierto es `Leontief Matrix def`.

El cuadrado grande cuyo contenido es `MATRIXINVERT(Matriz IA def[SecReg,SecReg1] )` es donde se programa el comportamiento del nodo, utilizando funciones definidas por Vensim o añadidas por el DLL.

`MATRIXINVERT()` es el nombre de la función de inversión matricial añadida por el DLL. Todas las funciones han de estar en mayúsculas. Todo lo que está entre paréntesis es la forma de referirse a la matriz que se le pasa como parámetro. En este caso esa matriz proviene de la variable cuya flecha apunta al nodo abierto.

El apartado `Functions` muestra la lista de funciones que se pueden utilizar. Se pueden filtrar de acuerdo a varios criterios, los cuales se pueden ver en el menú que está desplegado. El filtro aplicado es `User Defined`, que muestra las funciones añadidas a Vensim a través del DLL.

Además de tener su interfaz gráfica que facilita la comprensión del modelo y la creación de este, Vensim pone a nuestra disposición un modo de texto, donde se puede ver el diagrama como texto exclusivamente, y encontramos con detalle los comandos que luego compila Vensim para ejecutar el modelo.

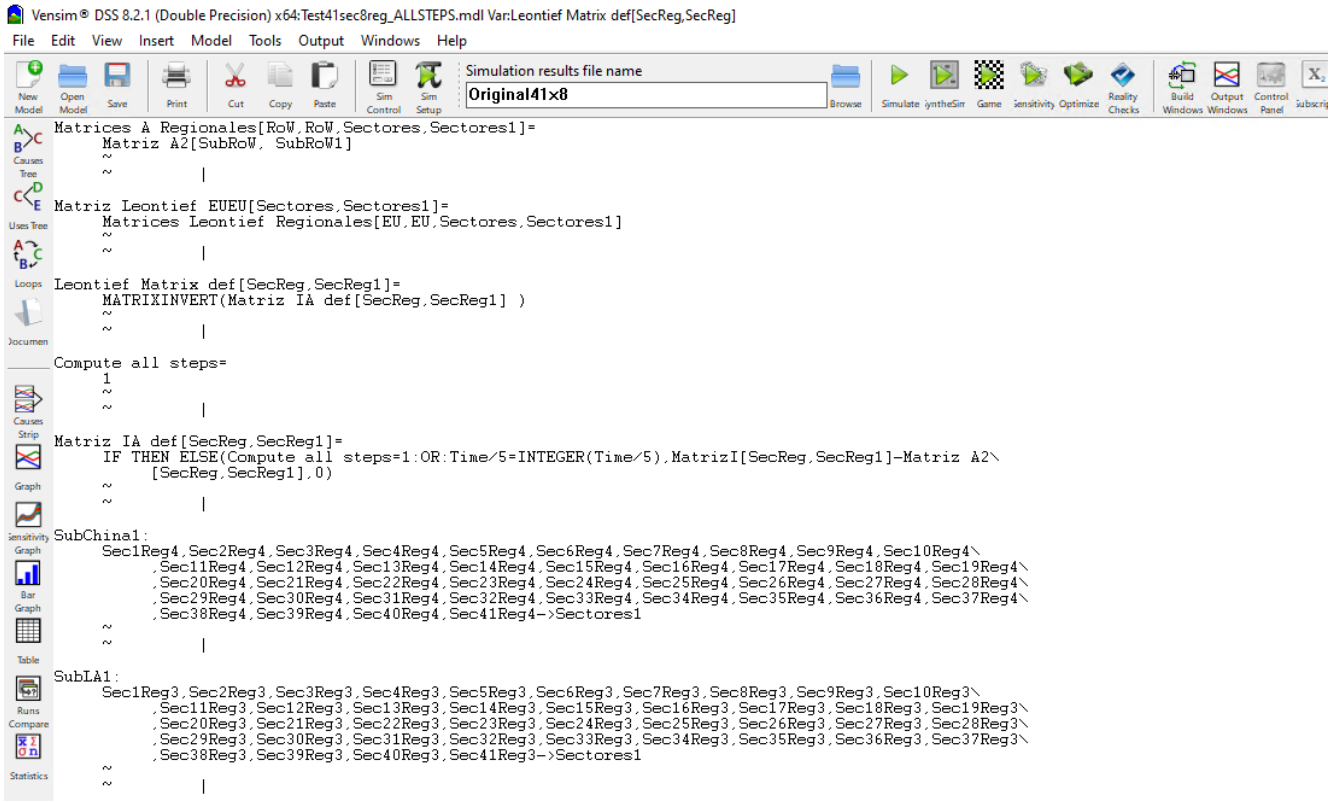


Figura 3.3: Modo de Vensim de edición de texto, incluyendo el código correspondiente al nodo `Matrices Leontief Regionales`

Este último modo ayuda a entender el funcionamiento interno del programa de una forma más concreta. Se puede ver el aspecto de este modo en la Figura 3.3.

3.2. Problemática planteada

Pese a que Vensim tiene mucha potencia a la hora de crear modelos complejos, Vensim no fue diseñado para realizar cálculos con grandes volúmenes de datos. Pese a ello, GEEDS se ha visto en la necesidad de incluir varias operaciones de inversión matricial en su proyecto, y han averiguado que esta operación es el mayor cuello de botella que tienen.

Teniendo en cuenta que las ejecuciones de los modelos llevan decenas de minutos u horas según el modelo, ellos esperan que acelerar esta operación les permita acelerar la ejecución del modelo. Además, cada modelo incorpora un volumen de datos mucho mayor, ya que cada vez que añaden una nueva fila y columna a la matriz cuadrada de datos pertinentes a los países, el número de datos que contiene la matriz crece exponencialmente, por lo que esperan que esta operación suponga un problema cada vez mayor en próximas iteraciones del modelo.

Vensim tiene el problema de que las funciones que realizan operaciones matemáticas no están tan optimizadas como lo están en una librería de operaciones matemáticas, y menos aún para matrices tan grandes que de no ser recorridas correctamente pueden aumentar mucho las operaciones de acceso a memoria, que tienen un coste de tiempo muy alto.

Se pusieron en contacto con el soporte de Vensim por si existía alguna alternativa mejor para lo que estaban haciendo y les respondieron que se podría solucionar a través de una función externa en un DLL.

Sin embargo este DLL hay que basarlo en un ejemplo dado por Vensim, escrito completamente en C y tiene que seguir una serie de estándares que no están claramente explicados. Este código además está compuesto de varios archivos.

3.3. Problemas encontrados

Los principales problemas que encontraron a la hora de desarrollar el DLL por su cuenta fueron:

- Se requiere mucho conocimiento de programación en C dado a que no es un lenguaje de programación sencillo de utilizar si no tienes conocimientos suficientes.
- Se requiere mucho conocimiento de programación en general para el dll ya que usa estructuras de datos complejas y creadas por ellos mismos y es difícil entender el propósito de muchas secciones del código si no entiendes muchos fundamentos de programación.
- Se requiere además conocimiento acerca de Vensim. Este punto lo cumplían de sobra pero impedía que varios de los problemas o confusiones que tengan acerca del código pudieran ser resueltos por un informático que desconozca Vensim.
- Se requieren también conocimientos acerca de optimización, ya que no es suficiente con crear un dll que realize las operaciones que necesitan sino que además necesita funcionar mejor que Vensim.
- Se requiere además experiencia en depuración de código ya que en caso de haber alguna clase de error en el código, Vensim se cierra sin devolver mensaje o código de error, por lo que es necesario tener conocimientos y experiencia detectando y resolviendo errores de código.

3.4. Solución planteada: Visual Studio

La primera opción elegida para desarrollar un DLL fue Visual Studio debido a que es una herramienta desarrollada por Microsoft y la que se usa por defecto a la hora de desarrollar archivos tipo DLL.

Además, DLL es el nombre de la extensión para archivos de bibliotecas de enlace dinámico dado por Microsoft para el uso de estas en sus sistemas operativos, por lo que se esperaba que una herramienta creada por la misma compañía ofreciese la mejor herramienta para la creación de estos.

Sin embargo Visual Studio está planteada para generar DLLs que puedan ser usados indistintamente en cualquier máquina que tenga Windows instalado, por lo que no permite la compilación de código con comandos exclusivos para cierto hardware, ya que eso haría que no pudiese ser usado en todos los ordenadores.

Como se ha explicado en el Capítulo 2, OpenBlas se instala de forma optimizada para cada máquina, por lo que contiene bastantes instrucciones específicas para el hardware en el que ha sido instalado incluso en los headers que se necesita importar para poder usar OpenBlas como DLL.

Tras no lograr resolver ni evitar este problema, hubo que cambiar el entorno de desarrollo a uno que sí permitiera la compilación de archivos DLL que contengan instrucciones exclusivas a cierto hardware.

Por suerte, había otro disponible para Windows que permitía realizar esto mismo porque de lo contrario hubiera sido necesario desarrollar el DLL en Linux y usar compilación cruzada para obtener el DLL, creando mayores dificultades para el testeo del código y más fuentes de posibles errores.

3.5. Solución planteada: Code::Blocks

Code::Blocks es un entorno de desarrollo gratuito de código abierto desarrollado por «The Code:Blocks team» muy similar a Visual Studio en apariencia y en funcionalidad, pero que permite mucha más libertad debido a que permite el uso de varios compiladores diferentes, el uso de plugins, y más opciones a la hora de decidir cómo compilar el código.

Sin embargo, la ventaja que más relevante resulta para el proyecto es que no tiene ningún problema para compilar código que contiene instrucciones optimizadas solo para cierto tipo de hardware.

3.6. Análisis del código presentado por Vensim

El código dado por Vensim como ejemplo para desarrollar un DLL es un ejemplo autodocumentado que cumple también la función de manual. Sin embargo, sigue siendo un ejemplo autodocumentado, por lo que no todo estaba claro y era necesario entender muchas partes para entender el código, por lo que aprender los contenidos de este ejemplo resultó ser un proceso iterativo que se extendió durante toda la duración del proyecto.

El código dado lo podemos dividir en varias secciones:

Interfaz del DLL para Vensim

Esta sección comprende las funciones que el DLL requiere tener. Cuando se le indica a Vensim a través de su configuración que cargue un archivo DLL, Vensim llama a una serie de funciones que han de estar disponibles y bien configuradas para asegurarse de que el DLL es compatible con Vensim y para recibir cierta información necesaria.

Esta información incluye los datos pertinentes a las funciones desarrolladas por el usuario. Dentro del código del DLL se ha de tener una estructura de datos creada que Vensim puede leer para saber:

- El nombre de cada una de las funciones creadas.
- Las indicaciones que ha de dar al usuario de Vensim acerca de los parámetros que la función recibe.
- El número de parámetros que recibe la función.
- Si la función devuelve un entero, un vector o una matriz.

- El código de función correspondiente a la función creada.
- Si la función modificará alguno de los parámetros dados.
- Una flag para indicar si la función devuelve exclusivamente un mensaje de texto o una serie de datos sin recibir parámetros o no.

Además, existe una función llamada `vensim_external` que es a la que Vensim llama cuando ejecuta una función del DLL. Esta función tiene dos parámetros, el código de la función a la que Vensim ha llamado, una estructura de datos que contiene todos los parámetros con los que se ha llamado a la función del DLL y el número de parámetros pasados.

Dentro de esta función se encuentra un switch que lee el código de la operación, y dentro de este se ha de llamar a la función correspondiente dentro del DLL.

```
switch(funcid)
{
    case INVERT_FUNC :
        rval = MATRIXINVERT(val[0].vec, val[1].vec) ;
        break ;
}
```

Cuadro 3.1: Extracto de código C del *switch* de la función `vensim_external`.

En este ejemplo se puede ver que el switch toma como parámetro el id de la función. En el caso de que este parámetro sea el correspondiente al id de la función de inversión matricial, se llama a la función de inversión matricial interna del DLL.

La función de inversión matricial solo tiene un parámetro, la matriz, ya que esta viene dada en una estructura de datos de la que podemos obtener toda la información relevante a esta. Sin embargo, Vensim no permite devolver el resultado a través de un `return`, sino que para devolver el resultado Vensim añade un parámetro al principio del array de parámetros `val` que se ha de modificar para que tenga los valores que se desean devolver.

`val` es el nombre del array con todos los parámetros que recibe el DLL. Este array está compuesto de elementos tipo union, de manera que hay que indicar el tipo del parámetro que se está recibiendo. Los principales tipos que puede tomar cada elemento de `vec` son:

- `val` para argumentos de tipo doble. Vensim utiliza siempre dobles, no enteros ni floats, así que en caso de querer utilizar un entero o un float se ha de utilizar un doble igualmente. Estos se pasan como valor. Vensim llama a este tipo de dobles `COMPREAL`.
- `vec` para vectores o matrices. Estos se pasan por dirección, así que hay que tener cuidado con modificarlos. Además, si se modifican sin haber marcado en la declaración de la función que se van a modificar argumentos, Vensim se cierra al ejecutar el modelo.
- `literal` para cadenas de caracteres.

Gestión de memoria

Esta sección comprende únicamente tres funciones, que son las usadas por el DLL para llevar un registro de toda la memoria que se ha reservado para poder limpiarla entre ejecuciones entre modelos.

Estas tres funciones son:

- `vext_allocate`: Esta función se encarga de la reserva de memoria.
- `vext_reallocate`: Esta función se encarga de modificar el tamaño de una zona de memoria reservada.
- `vext_clearmem`: Esta función borra toda la memoria reservada a través de las dos anteriores. Se ejecuta automáticamente cuando finaliza la ejecución de un modelo.

Además existen dos funciones, una que Vensim ejecuta al iniciarse la ejecución del modelo y otra que ejecuta al finalizarse. Estas funciones son: `simulation_setup` y `simulation_shutdown`.

Hay que asegurarse de inicializar en `simulation_setup` todas las variables estáticas globales que se necesiten.

`simulation_shutdown` se encarga de llamar a `vext_clearmem`.

Código de usuario

Esta sección comprende el resto del código desarrollado. Esto incluye tanto a las funciones a las que la función `vensim_external` llama directamente como a aquellas auxiliares que se usen.

3.7. Casos de prueba

Los modelos usados como casos de prueba para la matriz de inversión son tres versiones diferentes del mismo modelo, las cuales se diferencian en el tamaño de la matriz que cargan en el sistema. Estos modelos fueron aportados por el grupo GEEDS, quienes ya habían intentado resolver este problema y para ello habían creado estos modelos.

Estos modelos dados consisten en esencia en una carga de una matriz cuadrada de 78.400, 107.584 y 246.016 elementos respectivamente de un Excel, añadir un factor de aleatoriedad, invertir la matriz y separarla en matrices menores para obtener el formato que usarían en un proyecto real. Esta secuencia de acciones comprende un ciclo, y cada simulación de este modelo realiza 3200 de estos ciclos. En el Cuadro 3.2 se indica la relación de archivos utilizados para la prueba.

El nombre del modelo indica el tamaño de una de las dimensiones de la matriz cuadrada, y está compuesto de dos números que se han de multiplicar para hallar el tamaño de esta dimensión. Como es cuadrada, el número de elementos de la matriz se obtiene hallando el cuadrado del número de una de sus dimensiones.

Nombre de archivo	: Número de elementos
`Test35sec8reg.mdl` :	$78,400$
`Test41sec8reg.mdl` :	$107,584$
`Test62sec8reg.mdl` :	$246,016$

Cuadro 3.2: Correspondencia entre los nombre y el número de elementos de los casos de prueba.

Table | Matrices Leontief Regionales

File View Windows

Time (Time)	0	1	2	3	4	5	6	7	8	9	10	11	12
Matrices Leontief Regionales[EU,EU,sec1,sec1] : STOCK	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.1220
Matrices Leontief Regionales[EU,EU,sec1,sec1] : Original41x8	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.122015	0.1220
Matrices Leontief Regionales[EU,EU,sec1,sec2] : STOCK	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.2299
Matrices Leontief Regionales[EU,EU,sec1,sec2] : Original41x8	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.229926	0.2299
Matrices Leontief Regionales[EU,EU,sec1,sec3] : STOCK	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.3106
Matrices Leontief Regionales[EU,EU,sec1,sec3] : Original41x8	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.310603	0.3106
Matrices Leontief Regionales[EU,EU,sec1,sec4] : STOCK	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.2038
Matrices Leontief Regionales[EU,EU,sec1,sec4] : Original41x8	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.203845	0.2038
Matrices Leontief Regionales[EU,EU,sec1,sec5] : STOCK	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.07662
Matrices Leontief Regionales[EU,EU,sec1,sec5] : Original41x8	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.0766248	0.07662
Matrices Leontief Regionales[EU,EU,sec1,sec6] : STOCK	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.08169
Matrices Leontief Regionales[EU,EU,sec1,sec6] : Original41x8	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.0816907	-0.08169
Matrices Leontief Regionales[EU,EU,sec1,sec7] : STOCK	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.1729
Matrices Leontief Regionales[EU,EU,sec1,sec7] : Original41x8	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.172978	0.1729
Matrices Leontief Regionales[EU,EU,sec1,sec8] : STOCK	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.06562
Matrices Leontief Regionales[EU,EU,sec1,sec8] : Original41x8	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.0656263	-0.06562
Matrices Leontief Regionales[EU,EU,sec1,sec9] : STOCK	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.1521
Matrices Leontief Regionales[EU,EU,sec1,sec9] : Original41x8	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.152114	-0.1521
Matrices Leontief Regionales[EU,EU,sec1,sec10] : STOCK	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.07761
Matrices Leontief Regionales[EU,EU,sec1,sec10] : Original41x8	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.0776106	0.07761
Matrices Leontief Regionales[EU,EU,sec1,sec11] : STOCK	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.003948
Matrices Leontief Regionales[EU,EU,sec1,sec11] : Original41x8	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.00394833	0.003948
Matrices Leontief Regionales[EU,EU,sec1,sec12] : STOCK	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.00439
Matrices Leontief Regionales[EU,EU,sec1,sec12] : Original41x8	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.0043929	-0.00439
Matrices Leontief Regionales[EU,EU,sec1,sec13] : STOCK	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.004890
Matrices Leontief Regionales[EU,EU,sec1,sec13] : Original41x8	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.00489021	-0.004890
Matrices Leontief Regionales[EU,EU,sec1,sec14] : STOCK	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.1060
Matrices Leontief Regionales[EU,EU,sec1,sec14] : Original41x8	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.106026	0.1060
Matrices Leontief Regionales[EU,EU,sec1,sec15] : STOCK	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.1737
Matrices Leontief Regionales[EU,EU,sec1,sec15] : Original41x8	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.173762	-0.1737
Matrices Leontief Regionales[EU,EU,sec1,sec16] : STOCK	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.1811
Matrices Leontief Regionales[EU,EU,sec1,sec16] : Original41x8	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.181123	0.1811
Matrices Leontief Regionales[EU,EU,sec1,sec17] : STOCK	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.05091
Matrices Leontief Regionales[EU,EU,sec1,sec17] : Original41x8	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.0509119	-0.05091
Matrices Leontief Regionales[EU,EU,sec1,sec18] : STOCK	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.1058
Matrices Leontief Regionales[EU,EU,sec1,sec18] : Original41x8	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.105829	0.1058
Matrices Leontief Regionales[EU,EU,sec1,sec19] : STOCK	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.02529
Matrices Leontief Regionales[EU,EU,sec1,sec19] : Original41x8	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.0252931	-0.02529
Matrices Leontief Regionales[EU,EU,sec1,sec20] : STOCK	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.28798	0.287

Figura 3.4: Ejemplo de comparación de los resultados de dos ejecuciones. En la primera columna figura los instantes de ambas matrices (original y STOCK) y en las siguientes los valores en cada instante de tiempo.

El mayor problema que tienen estos modelos originales es que al tener un factor de aleatoriedad no se puede comprobar si el DLL devuelve los mismos resultados que devolvería Vensim, por lo que lo primero que se hizo fue eliminar este factor de aleatoriedad para que todas las ejecuciones devolvieran el mismo resultado.

Vensim permite guardar los resultados de las ejecuciones y cargar varios en el programa para compararlos usando la herramienta del programa «Table» sobre la variable que almacena los resultados obtenidos de la inversión matricial.

Esta herramienta muestra los resultados almacenados en esa variable en cada paso transcurrido en la ejecución obtenidos en cada una de las ejecuciones cuyos datos se hayan cargado en el programa.

Esta figura muestra el resultado de usar la herramienta «Table» con dos sets de datos cargados en Vensim sobre el nodo `Matrices Leontief Regionales`. «STOCK» es el nombre del resultado de una de las ejecuciones, y es la ejecución más reciente realizada. El nombre «STOCK» es el dado por defecto por el script usado para ejecutar el modelo.

Este set de datos se está comparando con el «Original41x8», y se puede comprobar que ha devuelto los resultados esperados comparando los valores que toman las casillas por parejas. Este es el método

usado para confirmar que el DLL funciona como se espera.

Estos datos se cargan en el programa con el uso de la herramienta «Control Panel» que te permite cargar los archivos de resultados que se encuentren en la misma carpeta donde esté guardado el modelo. Puedes cargar cualquier número de ejecuciones, incluso cero, pero no ha sido necesario nunca cargar más de dos.

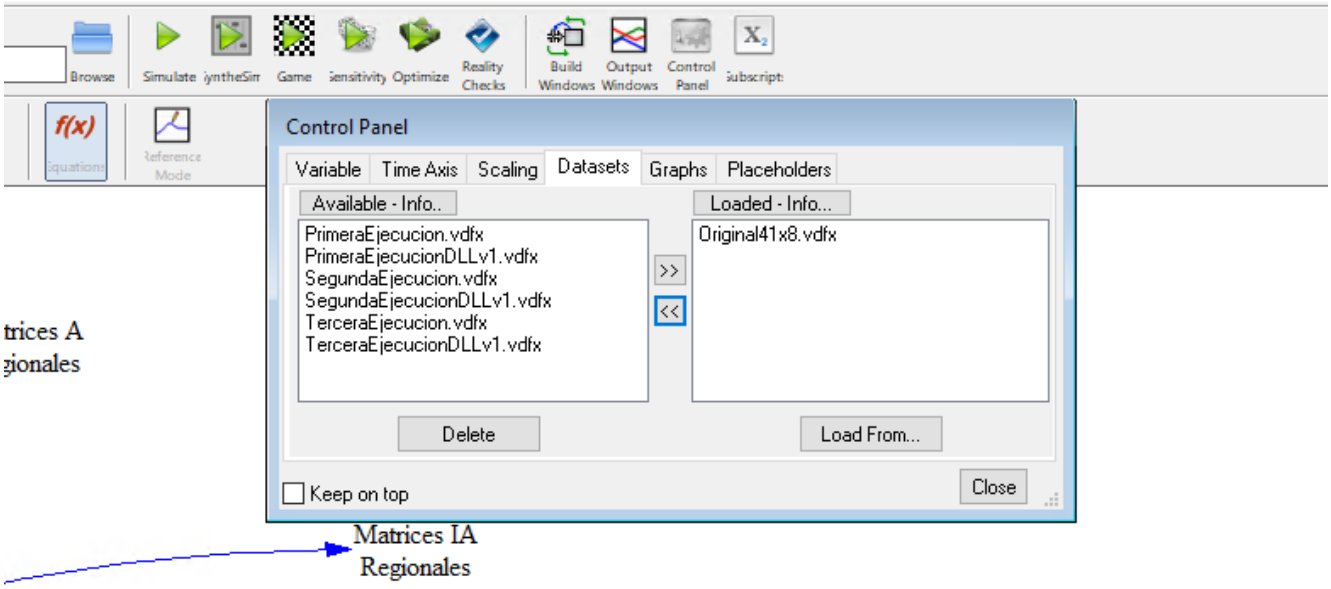


Figura 3.5: Resultado de abrir la herramienta «Control Panel» en uno de los modelos

En esta figura se puede observar el resultado de abrir la herramienta «Control Panel». Hay dos columnas, en la de la izquierda se muestran los archivos con datos detectados en la carpeta del modelo y a la derecha los datos cargados en el programa.

Como se puede ver, están cargados en Vensim los sets de datos llamados «STOCK» y «Original41x8»

Además, se usó un pequeño script dado también por el grupo GEEDS para guardar la cantidad de tiempo que tarda una simulación en completarse. Este script almacena en un archivo de texto la hora en la que empezó finalizó la ejecución, junto a los minutos y los segundos.

Teniendo esto en cuenta, ya se disponían de herramientas para comprobar que el DLL devolvía los resultados esperados y la diferencia de tiempo entre usar el DLL y no usarlo.

3.8. Descripción del prototipo 1

El primer prototipo es sencillo y consiste en adaptar el ejemplo dado por Vensim a código que pueda ser compilado por Code::Blocks y cuyo DLL resultante pudiera ser leído y ejecutado por Vensim.

Para esto hubo principalmente que modificar el nombre y los includes de varios de los archivos para que coincidan con el árbol de directorios.

Este prototipo formó la base para prototipos posteriores

3.9. Pruebas realizadas

Se realizaron dos pruebas sencillas para comprobar el correcto funcionamiento de este prototipo.

La primera consiste en cargar el DLL obtenido en Vensim y que este lo cargue sin errores, y que además lo cargue automáticamente cada vez que se abre el programa. Esta prueba nos indica tanto que se han corregido todos los errores del ejemplo dado por Vensim como que no hay problemas en las funciones pertinentes a la comunicación inicial de Vensim.

Además, para comprobar si el DLL ha sido cargado correctamente, se abre el menú desplegable que enumera las funciones que añade este DLL, por lo que al ver que la lista de funciones mostradas es correcta, se comprueba que la comunicación inicial entre el DLL y Vensim ha sucedido como se esperaba.

La segunda prueba comprende sustituir en un ejemplo la función de inversión de matrices de Vensim por la cargada en el DLL de ejemplo de Vensim y comprobar que, pese a que como se esperaba tarda más tiempo, los resultados obtenidos en la simulación son los esperados, idénticos a los originales.

Capítulo 4

Integración de Vensim y OpenBlas

4.1. Solución adoptada

A la hora de elegir las versiones del software a utilizar se eligieron las más nuevas disponibles en el momento. Las versiones utilizadas han sido:

- **Code::Blocks** versión 20.03mingw [15]
- **OpenBlas** versión 3.15 [9]
- **Miniconda 3** versión Python 3.9 [16]
- **Visual Studio** versión 2019 [14]
- **Vensim** versión DSS 8.2.1.

Debido a que OpenBlas se compila como una biblioteca de 64 bits, el dll creado se compila como una biblioteca de 64 bits también, por lo que se requiere de Vensim 8 o superior para poder ser usada ya que es la versión a partir de la cual se pueden cargar en Vensim bibliotecas de 64 bits.

También es posible que versiones posteriores a la 8 incluyan elementos que hagan que el DLL sea incompatible. Por suerte cada nueva versión de Vensim DSS trae la versión actualizada del DLL de ejemplo y solo debería de hacer falta sustituír ciertos archivos por otros nuevos, sin modificar el código en el archivo `venext.c` donde se incluye todo el código escrito en este trabajo.

El directorio por defecto donde se puede encontrar el ejemplo del código del DLL tras instalar Vensim es:
C:\Users\Public\Vensim\COMP\VensimExternalFunctionLibrary\Windows\VensimExternalFunctionLibrary

4.2. Análisis inicial de rendimiento

Para el análisis de rendimiento se usaron los casos de prueba explicados en el Capítulo (). Como las ejecuciones resultaron tardar entre 5 y 20 minutos y el tiempo de ejecución varía bastante dependiendo de la máquina en la

que se ejecute, se buscó obtener una referencia de tiempo meramente orientativa, por lo que se ejecutó cada caso de prueba 3 veces para realizar la media.

Estas medidas se tomaron de una ejecución en una máquina virtual con dos cores disponibles.

Más adelante se tomarían medidas de la misma manera de Vensim usando el DLL para comprobar si existe una mejoría de tiempo y qué nivel de mejora se puede esperar en los modelos utilizados.

Tamaño de fila	Número de elementos	Orden de ejecución	Tiempo de ejecución	media
35x8	78400	1º	$265s$	$264s$
		2º	$262s$	
		3º	$266s$	
41x8	107584	1º	$402s$	$397s$
		2º	$392s$	
		3º	$399s$	
62x8	246016	1º	$1267s$	$1249s$
		2º	$1222s$	
		3º	$1258s$	

Cuadro 4.1: Medidas de referencia obtenidas al ejecutar 3 veces cada caso de prueba con Vensim sin DLL. Estos contienen matrices de diferentes tamaños indicados en «Número de elementos»

4.2.1. Descripción del prototipo 2

Este prototipo consiste en una terminal que muestre los resultados de realizar una operación de OpenBlas sobre una matriz pequeña. El objetivo de este prototipo es obtener un entorno de pruebas para lograr entender bien cómo operar usando OpenBlas y poder comprobar de forma rápida y directa que funciona correctamente.

Más adelante, en caso de tener dudas acerca de si se está llamando adecuadamente a una función o se desea realizar pruebas acerca de modificar una variable de la función se usará este prototipo para comprobar que tal función nueva a incorporar en el código del DLL funciona como se espera que haga.

Como la primera operación que se desea realizar usando OpenBlas es la inversión matricial, esta es la operación que usará este prototipo. Para realizar esta operación, se requiere de la llamada de dos funciones de OpenBlas.

Este prototipo tiene una matriz cuyos valores se establecen en el código, una llamada a una función en la que se opera sobre esta matriz y una creación de una terminal sobre la cual se representan los valores finales que tiene esta matriz al acabar la ejecución del programa.

4.2.2. Pruebas realizadas

La prueba realizada sobre este prototipo es establecer una matriz de 5x5 con valores no aleatorios y contrastar que los valores mostrados en la terminal corresponden con los obtenidos realizando la inversión de la misma matriz usando Wolfram Alpha.

Esta prueba se realizó tres veces con valores diferentes de la matriz inicial.

Capítulo 5

Análisis y Diseño de la biblioteca presentada

En este capítulo se explica el funcionamiento y el desarrollo de las funciones desde el análisis de los requisitos de estas funciones hasta el desarrollo y las pruebas realizadas.

5.1. Requisitos de la interfaz de programación. Casos de uso

Para simplificar el uso de las funciones externas, se propuso como objetivo que la llamada a la función y su resultado se acercaran lo máximo posible a cómo se utilizan funciones similares en Vensim.

Además, al utilizar una función Vensim te permite mostrar cuántos parámetros le has de pasar a esta junto a el nombre que tiene cada uno.

Estaba planeado desarrollar dos funciones que supusieran una mejora directa respecto a funciones similares de Vensim: Inversión matricial y la operación SUM.

La función de inversión matricial realiza, como su propio nombre indica, la inversión de una matriz. La versión de Vensim llamada `INVERT MATRIX` recibe dos argumentos: la matriz y su tamaño. Sin embargo como Vensim ya le pasa esa información al DLL por medio de la estructura de datos que encapsula a la matriz pasada como argumento, la función desarrollada en el DLL solo tendrá un parámetro, la matriz. Y para conservar un nombre similar, su nombre es `MATRIXINVERT`.

La función SUM tiene un funcionamiento muy complejo en Vensim que es imposible de reproducir a través de un DLL de funciones externas. Sin embargo, el uso que exige la mayor cantidad de tiempo en los modelos del grupo GEEDS es bastante concreto. Juntan varias matrices creando una matriz enorme de muchas dimensiones y quieren sumar los datos por filas, columnas u otras dimensiones.

Además, se planeó crear una función para añadir una matriz al espacio de memoria del DLL, de manera que esta función llamada `ADDMATRIX` toma como argumento una matriz y un nombre para esta y la guarda en la memoria del DLL.

Esto es complementario a una función para recuperar esa matriz del espacio de memoria del DLL llamada `GETMATRIX` que solo toma como argumento el nombre de la matriz que se quiere recuperar y devuelve la matriz en cuestión.

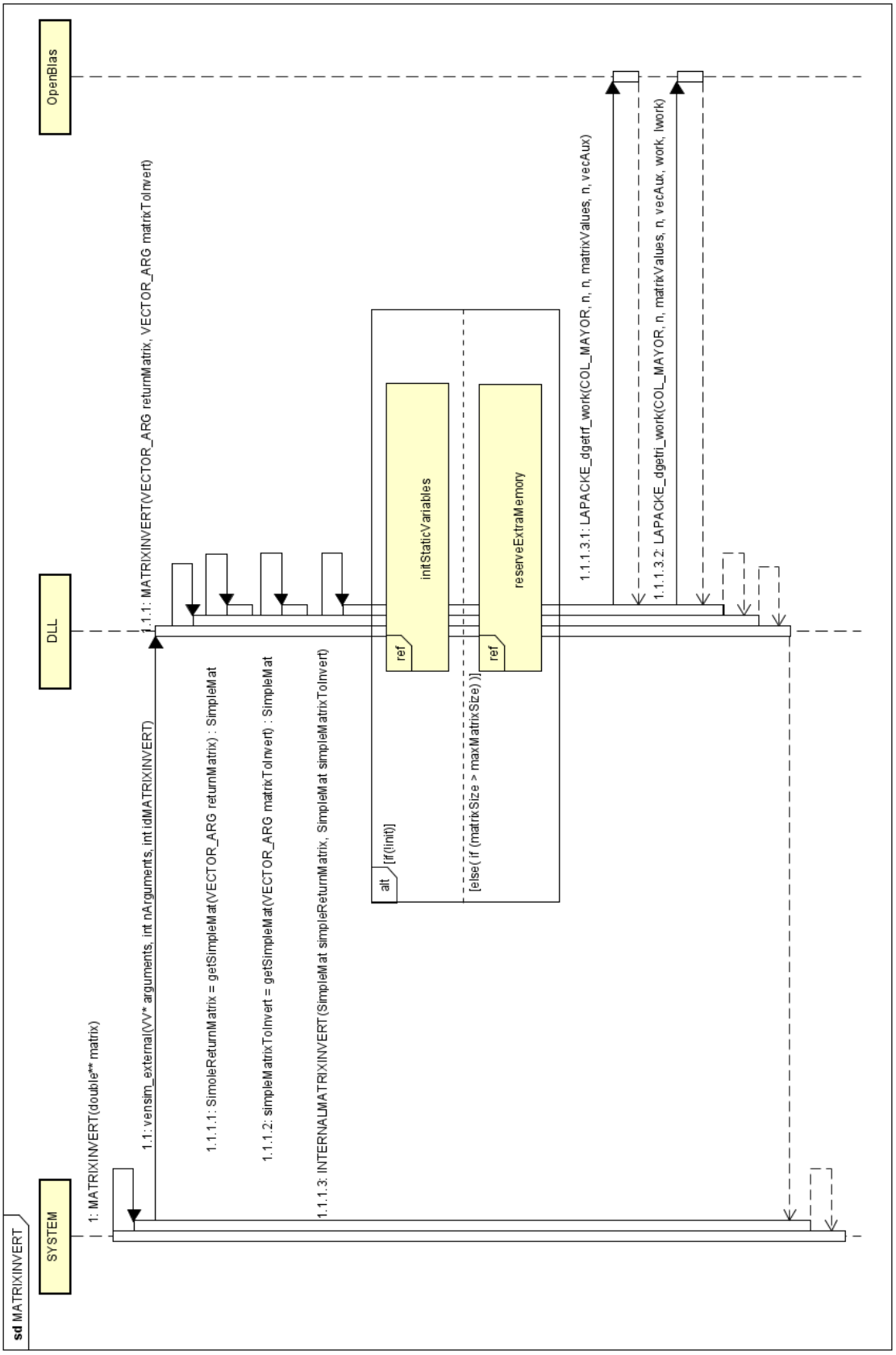


Figura 5.1: Diagrama de secuencia para la función MATRIXINVERT.

Por último, la operación DELMATRIX para el borrado de una matriz que esté en el espacio de memoria del DLL.

Además, por cada función desarrollada, habrá una función complementaria que reciba como argumento el nombre de la matriz con la matriz de trabajo almacenada y opere sobre la memoria del DLL. Por ejemplo, la función COMPUTEMATRIXINVERT toma como argumento el nombre de la matriz guardada en el DLL y la invierte sin necesidad de proporcionarsela como argumento.

5.2. Diseño de la solución

Para poder entender bien cómo se comunica Vensim con el DLL se va a explicar en profundidad el diseño y el funcionamiento de la función MATRIXINVERT con ayuda de la Figura 5.1, en la que se muestra el diagrama de secuencia de esta función.

Los detalles de la comunicación inicial entre Vensim y el DLL se pueden encontrar en el Capítulo 3.

Cuando Vensim se encuentra en el modelo una operación perteneciente al DLL, Vensim encapsula todos los argumentos junto al objeto que se usará para devolver la respuesta a Vensim en un array, es decir, este array tendrá un tamaño igual al número de argumentos pasados mas uno.

Este vector es el primer argumento de la función vensim_external, que es la función a la que Vensim llama cada vez que se encuentra una operación del DLL en su modelo. El segundo argumento de esta función es el número de argumentos y el tercero es un integer correspondiente al código que identifica la operación que se quiere realizar.

Dentro de la función vensim_external se encuentra el switch explicado en el Capítulo 3 que se encarga de llamar a la función de inversión matricial del DLL. Estos argumentos se los pasa como un objeto tipo VECTOR_ARG, que es el objeto con el que Vensim encapsula las matrices y contiene datos útiles como el número de dimensiones que tiene y el tamaño de cada una de ellas.

Sin embargo, para que la función de inversión matricial sea compatible con el formato de cálculo con matrices almacenadas en el DLL, la función interna que realiza el cálculo de inversión matricial tiene como parámetros dos matrices de tipo SimpleMat, el tipo de objeto con el que se almacenan las matrices en el DLL.

Por esa razón esta función llamada MATRIXINVERT transforma las matrices de VECTOR_ARG a SimpleMat a través de la función del DLL getSimpleMat.

A continuación se llama desde el DLL a la función INTERNALMATRIXINVERT la cual es la que se encarga de llamar a OpenBlas para realizar los cálculos. Esta función realiza la operación de inversión matricial y guarda el resultado en el argumento simpleReturnMatrix cuyos datos serán pasados de nuevo al argumento returnMatrix una vez haya finalizado la ejecución de la función INTERNALMATRIXINVERT y la función MATRIXINVERT recupere el control. Este argumento será el que lea Vensim para obtener la matriz devuelta por la función. Los detalles del funcionamiento de INTERNALMATRIXINVERT serán explicados en la Sección 5.3.

Vensim realiza comprobaciones de que el número de elementos de la matriz devuelta coincida con los que espera recibir, es decir, que coincida con los indicados en la estructura de datos VECTOR_ARG que envuelve la matriz. Vensim sabe qué resultado esperar por las indicaciones que el desarrollador del modelo ha de dar.

Además hay que tener en cuenta que este diagrama de secuencia coincide con la última iteración realizada sobre la función de inversión matricial, que se realizó durante el desarrollo del prototipo 4. La descripción del prototipo 3 comprende solo la función que en este diagrama se llama INTERNALMATRIXINVERT.

5.3. Descripción del prototipo 3

Este prototipo combina los resultados obtenidos en los dos prototipos anteriores, creando una función semejante a la dada por Vensim, pero cambiando completamente el código y modificando las variables que se pasan.

Para lograr esto, hubo que adaptar la matriz tal y como la lee Vensim (tal y como se explica en el Capítulo 3) a un formato que pueda ser usado con OpenBlas, es decir, un array de doubles junto a la cantidad de elementos que este tiene. Además, hay que inicializar correctamente las otras variables que OpenBlas requiere para funcionar correctamente.

OpenBlas ofrece dos versiones de cada función: una en la que OpenBlas se encarga de toda la gestión de memoria extra que necesiten las funciones y otra en la que el usuario le da el espacio de memoria extra para operar.

Como Vensim tiene sus propias funciones para reservar o borrar memoria, las cuales se explican con más detalle en el Capítulo 3. Es necesario usar las versiones de las funciones de OpenBlas en las que el usuario da a OpenBlas la memoria extra a usar, ya que se requiere reservar esta memoria extra usando las funciones de Vensim para asegurar su correcto funcionamiento.

Por suerte estas funciones de OpenBlas se pueden llamar de una forma en la que devuelve el espacio de memoria ideal para realizar la operación eficientemente. Tarda más si este espacio es mucho mayor del necesario, pero la operación para averiguar el espacio óptimo también tiene un tiempo de ejecución a tener en cuenta.

Esta memoria reservada se guarda en variables estáticas para evitar tener que reservar memoria cada vez que se llame a esta función. Por eso la variable estática `Matrix_invert_maxn` contiene el número de elementos de la fila o columna de la matriz más grande que se ha recibido. En este punto ya se han realizado comprobaciones de que la matriz recibida es cuadrada. En caso de que esta variable no haya sido inicializada, se reserva la memoria original para todas las variables estáticas a través de la función de reserva de memoria del DLL.

En caso de que se reciba una matriz con más elementos que ninguna recibida anteriormente, se utilizan las funciones de memoria del DLL para reservar más memoria para los elementos estáticos que la necesiten.

La razón por la que se realizaron casi todas las preparaciones para el desarrollo de la funcionalidad de inversión matricial durante el desarrollo del prototipo de utilizar llamadas vacías a OpenBlas para comprobar si Venism permite este tipo de comunicaciones es que hubo que aprender a utilizar las llamadas de gestión de memoria dadas por Vensim.

Una vez se tuvo la necesidad de aprender a utilizar estas funciones de reserva de memoria y reservar memoria apropiadamente, se decidió crear reservas de memoria que pudieran ser utilizadas por la función final, por lo que hubo que aprender bien acerca del paso de argumentos en Vensim y acerca de las estructuras de datos.

Además hubo que cambiar las llamadas a OpenBlas a las versiones en las que te encargas como programador de reservar la memoria extra que necesite para trabajar y se decidió crear estas reservas de memoria de forma que fueras útiles para la inversión matricial.

5.4. Pruebas realizadas

Para comprobar su funcionamiento se ejecutaron los casos de prueba explicados en el Capítulo 3, confirmando que los datos devueltos son correctos. Se realizaron tres ejecuciones de cada caso de prueba y se obtuvieron los siguientes resultados:

Tamaño de fila	Número de elementos	Orden de ejecución	Tiempo base	Media base	Tiempo con DLL	Media con DLL	Diferencia	Mejora
		1º	$265s$		$219s$			
`35x8`	78400	2º	$262s$	264	$222s$	$227s$	$-45s$	$17,15\,\%$
		3º	$266s$		$241s$			
		1º	$402s$		$290s$			
`41x8`	107584	2º	$392s$	397	$290s$	$289s$	$-107s$	$27,07\,\%$
		3º	$399s$		$288s$			
		1º	$1267s$		$626s$			
`62x8`	246016	2º	$1222s$	1249	$640s$	$637s$	$-623s$	$49,88\,\%$
		3º	$1258s$		$645s$			

Cuadro 5.1: Comparación de los tiempos de ejecución de los casos de prueba con Vensim con DLL para diferentes tamaños de matrices con las referencias tomadas en la Tabla 4.1. En la última columna se muestra la mejora porcentual obtenida; puede verse que para matrices cada vez mayores crece la mejora.

En esta Figura 5.2 se puede apreciar cómo la mejora que aporta el DLL es mayor cuanto más elementos tiene la matriz. El porcentaje de mejora concreto se puede ver en la tabla, y llega a ser una mejora de $49,88\,\%$ del tiempo en el caso de prueba con la matriz de mayor tamaño.

Viendo la tendencia que tiene, se espera una mejora mayor aún para matrices con más elementos. Las matrices que usan en el proyecto LOCOMOTION tienen un tamaño mayor que las del caso de uso más grande, y cada vez van añadiendo más elementos y se van haciendo más grandes.

Por esta razón se pensó que podría ser interesante estudiar la evolución que tiene la mejora de rendimiento que aporta el DLL, ya que aunque tengamos pocos datos de referencia, el tiempo que tardan las ejecuciones con Vensim parece escalar exponencialmente con el número de elementos de la matriz.

Es posible que el tiempo que tarde en ejecutarse con el DLL sea lineal o exponencial, y sería interesante averiguar la relación entre el escalado de tiempo con y sin DLL y para averiguarlo harían falta más casos de prueba con matrices más grandes.

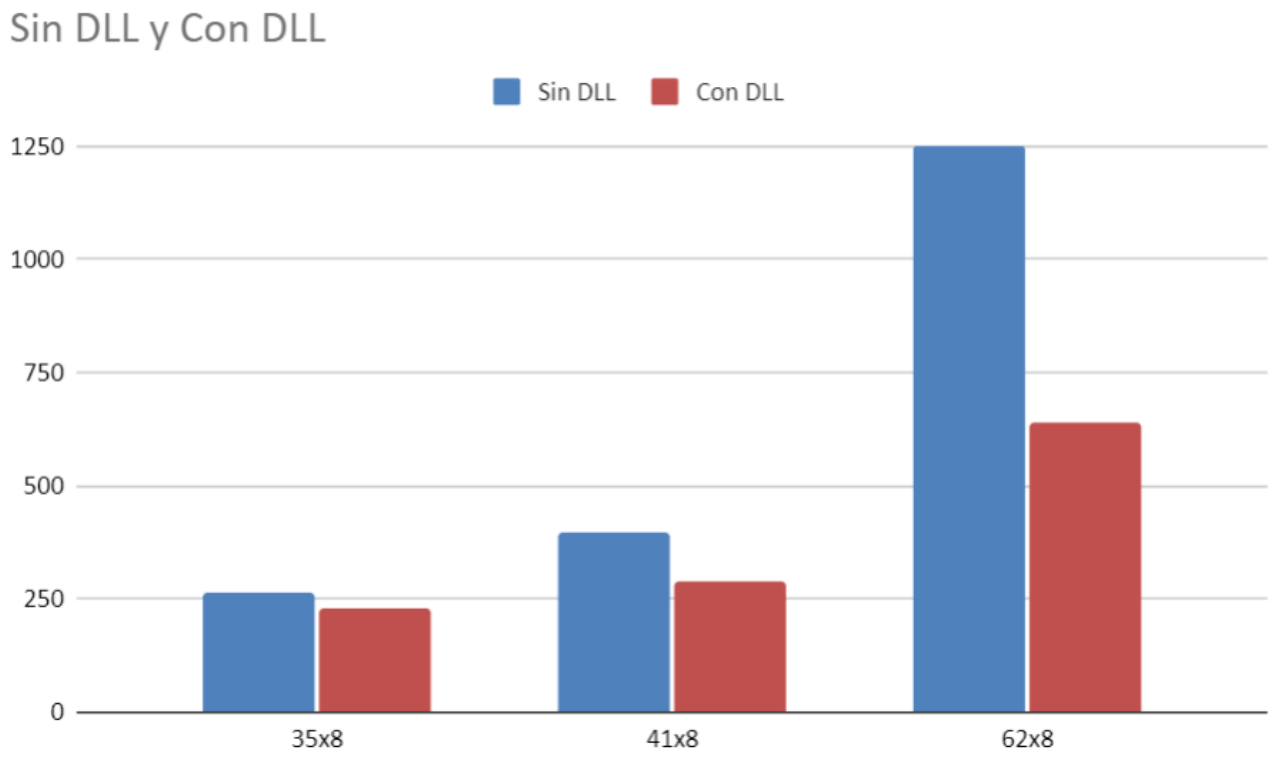


Figura 5.2: Gráfico con la media de tiempo de cada caso de prueba con y sin DLL correspondiente a los datos mostrados en la Tabla 5.1

Sin embargo, pese a que se ha estudiado suficiente acerca de la creación de modelos en Vensim para modificar los modelos existentes, no he logrado alcanzar el nivel para crear modelos complejos desde cero ni para modificar el modelo para incluir matrices más grandes.

Esto se debe a que las matrices se han de cargar de un excel a través de una operación que ya es poco intuitiva, pero principalmente debido a que Vensim tiene una serie de constantes que le indican a las funciones el tamaño de las matrices en los modelos de ejemplo. Esta herramienta es muy potente porque permite modificar mucho el modelo con solo cambiar las variables de entrada y estas constantes globales, las cuales se pueden modificar, pero necesitas entender muy bien tanto el funcionamiento de Vensim como el funcionamiento del modelo para poder cambiar estas constantes y que todo siga funcionando. Puede haber cientos de estas constantes que se generan a través de una serie de constantes iniciales, que son las que se pueden modificar y hay decenas de estas.

Debido a la alta complejidad de modificar estos modelos para escalar el tamaño de las matrices para un usuario sin experiencia anterior en Vensim no he sido capaz de crear estos modelos durante el tiempo disponible en el proyecto.

5.5. Descripción del prototipo 4

Se planteó una forma alternativa de uso de la función externa, con la cual se almacenan las matrices en el espacio de memoria del DLL en vez de en el propio Vensim.

Esto supone ofrecer al usuario la posibilidad de almacenar y nombrar estas matrices a través de una función para ser luego usadas en otras funciones. La ventaja que ofrece este método es que, en caso de querer realizar

varias operaciones en serie sobre una misma matriz, se puede limitar las transacciones de esos grandes volúmenes de datos entre Vensim y el DLL desde dos por operación a dos en total, el paso de la matriz inicial al DLL y la devolución del resultado al finalizar la cadena de operaciones.

Como no se deseaba saturar el archivo con la mayor parte del código del DLL con muchos objetos estáticos necesarios para almacenar todas las matrices que se desee almacenar junto a los datos relativos a estas matrices, se optó por crear una función que se encargue de realizar el guardado y el borrado de matrices y se pueda indicar la operación que se desea realizar mediante un código de operación que se indica como parámetro para la función.

Este método permite hasta devolver resultados intermedios a Vensim en caso de que se necesiten para otra cosa sin interrumpir la secuencia de operaciones que se quieren realizar sobre la matriz, dejando un total de dos transacciones por cadena de operaciones más una por cada resultado intermedio que se desee obtener.

Para ello, se necesitan realizar tres funciones extra y modificar la función de inversión matricial. Las funciones extra son: almacenar una matriz dándole un nombre, recuperar esa matriz y borrar esa matriz.

Las matrices se almacenan en el DLL en un objeto mucho más sencillo que los usados por Vensim. Este objeto llamado SimpleMat (matriz simplificada) almacena el array de valores, el número de dimensiones y el tamaño de cada dimensión.

Para poder llamar a la función de inversión matricial usando este objeto, se dividió esta función en una función interna privada con la mayor parte del código pero leyendo y devolviendo matrices en formato SimpleMat en lugar del usado por Vensim, y la función usada desde Vensim a la cual se llama de la misma forma que la versión descrita en el prototipo 3 y cuyo papel es transformar el objeto de matriz dado por Vensim a este nuevo objeto SimpleMat y llamar a la función de inversión matricial con este nuevo objeto. Al finalizar, transformar de nuevo el resultado devuelto en el formato de objeto SimpleMat a un objeto legible por Vensim.

Además, se creó una nueva función para su uso desde Vensim de inversión matricial cuyo parámetro en lugar de ser la matriz almacenada en Vensim es un string con el nombre de la matriz almacenada en el DLL. Esta función llama a la función de inversión matricial interna del DLL y guarda el resultado sobreescribiendo la matriz con la que se ha llamado a la función.

En caso de añadir más funciones a este DLL estas deberán seguir un formato similar a la de inversión matricial, es decir tener una versión privada donde se realizan las operaciones y dos formas diferentes de llamarla, una dando la matriz almacenada en Vensim y otra usando el nombre de una función almacenada.

Por esta razón se creó una función que dada una matriz en formato `VECTOR_ARG` te devuelve una matriz en formato `SimpleMat`. Esta función se llama `getSimpleMat`. Sin embargo, esto creó un problema, y es que para poder almacenar estas matrices en formato `SimpleMat` se ha de reservar memoria para todas sus variables y copiarlas, incluyendo todos los datos de la matriz.

Si tenemos en cuenta que hay que pasar todas las matrices a formato `SimpleMat`, esto supondría crear una copia de todas las matrices que se usaran en cualquier operación, incluso cuando son llamadas de forma normal. Esto aumenta el tiempo que llegaba a tardar la función `MATRIXINVERT` hasta casi el doble, por lo que se creó una nueva función llamada `getQuickSimpleMat`, la cual funciona exactamente igual a su similar `getSimpleMat`, solo que en vez de reservar memoria y copiar todos los elementos de la matriz, copia la dirección del puntero, de forma que se ahorra una enorme cantidad de tiempo, pero a cambio si se trata de almacenar matrices creadas por `getQuickSimpleMat` se crearán fallos de consistencia en los datos y acabará crasheando Vensim.

También se ha creado una función `freeSimpleMat`, a la cual le pasas como parámetro un objeto tipo `SimpleMat` y libera toda la memoria asociada al objeto. Sin embargo a esta función no se le pueden pasar objetos creados con la función `getQuickSimpleMat` o se intentaría borrar memoria de Vensim y dejaría de funcionar ya que el puntero de los datos apunta al espacio de memoria de Vensim.

Vensim al finalizar la ejecución de cada modelo ejecuta un `free()` a todos los objetos que han sido reservados a través de las funciones de reserva de memoria dadas por el DLL cuando llama la función `vext_clearmem`. Si no se utiliza la función `freeSimpleMat` para liberar memoria, la memoria que usa Vensim crece constantemente hasta saturar la memoria disponible, sin embargo si se utiliza, al finalizar la ejecución del modelo en la llamada a `vext_clearmem` se ejecuta `free()` sobre memoria ya liberada y crashea antes de guardar los datos.

Esto lleva al intento de que `getSimpleMat` reserve la memoria necesaria a través de `malloc`. Sin embargo, por razones que no he llegado a descubrir parece que al reservar memoria directamente luego Vensim tiene problemas al escribir los datos de la simulación en el archivo y se corrompen o crashea en el intento.

5.6. Pruebas realizadas

Las únicas pruebas realizadas exitosamente son las del uso de `getQuickSimpleMat`, función a la que se llama antes de llamar a la nueva `INTERNALMATRIXINVERT`. Todas las demás han producido errores que se han ido mencionando en la explicación del desarrollo del prototipo.

5.7. Descripción del prototipo 5

El prototipo 5 consiste en adaptar la función SUM de Vensim a una más eficiente pero más limitada. Esto se debe principalmente a que la función SUM es de las más versátiles y completas que ofrece Vensim.

El uso que se le da en los proyectos que ocupa la segunda mayor cantidad de tiempo después de la inversión matricial es uno muy concreto: sumar todos los elementos de una dimensión de una matriz reduciendo su número de dimensiones. Las dimensiones que quieren reducir pueden ser una o varias e incluso intercaladas.

Debido a que OpenBlas no viene con una función que realice esta operación concreta directamente, existen dos maneras de abordar el problema:

- Crear un algoritmo que limite que las dimensiones solo puedan ser colapsadas empezando desde la que está más a la derecha. Esto limitará al programador de los modelos de Vensim a acceder a la matriz de forma óptima, y es sencillo crear un algoritmo eficiente que realice la suma con un solo recorrido de la matriz.

  Esta limitación parece ser posible implementarla en los proyectos según conversaciones con integrantes de GEEDS, ya que esas matrices multidimensionales las crean ellos juntando muchas otras matrices más pequeñas.

  Se espera que al obligar a operar al usuario de Vensim a realizar esta operación de forma que los accesos de memoria sean eficientes y se pueda realizar un único recorrido de la matriz pueda ahorrar tiempo.

- Crear una función que realice una multiplicación matricial entre la matriz original y otra de dimensiones iguales pero cuyos elementos sean unos y ceros para obtener utilizando OpenBlas los resultados que deseamos pero de forma descolocada.

  Esto permitiria mantener la libertad de colapsar dimensiones de forma intercalada y utilizando OpenBlas se obtendría un muy buen rendimiento en la operación, sin embargo asegurarse de que funcione para cualquier combinación de dimensiones originales y a colapsar tal y como funciona en Vensim llevaría mucho tiempo de programar.

Como a estas alturas del proyecto ya se disponía de una cantidad limitada de tiempo, se optó por realizar la primera opción.

5.8. Pruebas realizadas

Tal y como se ha explicado en el la Sección 5.4 no he alcanzado un nivel suficiente en modelado en Vensim, por lo que no he podido crear casos de prueba para usar con la función SUM desarrollada.

Además, esta función tiene un problema adicional y es que la matriz que se recibe y la que se devuelve no tienen las mismas dimensiones, y es posible que esto pueda impedir la existencia de una sola versión de la función SUM, sino que haya que crear una versión para cada número de dimensiones que pueden ser devueltas (entre 1 y 7 dado que Vensim parece tener un límite de 8 dimensiones para las matrices).

Esto se debe a la existencia de una variable llamada `num_loop` que modifica la cantidad de las constantes de Vensim llamadas `subscripts` y es posible que esta variable obligue a que la función no pueda modificar el número de dimensiones de la matriz que puede devolver, es decir, que si está programada para devolver matrices de 3 dimensiones solo puede devolver matrices de 3 dimensiones.

Sin embargo es posible que si se programa la función para devolver matrices de 8 dimensiones se puedan devolver matrices de hasta 8 dimensiones ya que es el modelo de Vensim quien indica el número de dimensiones que tendrá la matriz de retorno.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

El objetivo principal de este proyecto, acelerar la inversión matricial en Vensim, ha sido cumplido con éxito. Tras el desarrollo de este trabajo se ha obtenido un DLL compilado que permite acelerar la operación de inversión matricial sobre todo para los casos más relevantes que son las matrices de gran tamaño y se ha obtenido también un manual de instalación de este DLL junto a OpenBlas.

Además, se ha investigado mucho acerca de librerías de álgebra lineal, de entornos de desarrollo para DLLs para Vensim y del funcionamiento del propio Vensim, y los resultados de esta investigación están presentes en esta memoria.

A partir de esos resultados obtenidos mas las bases creadas para ambos objetivos que no se han llegado a cumplir se puede extender este trabajo para cumplir con los objetivos que faltan y muchos otros que se mencionan más adelante.

En resumen, los objetivos cumplidos completamente han sido:

- Realizar un estudio de las librerías disponibles que permitan realizar ese tipo de cálculos.
- Realizar un estudio de la tecnología disponible para la creación de DLLs.
- Crear un DLL que permita añadir funciones externas a Vensim.
- Desarrollar una función externa que permita realizar la operación de inversión matricial en menor tiempo que Vensim para matrices grandes.
- Redactar un manual de instalación para la librería de cálculo utilizada.
- Redactar un manual de uso de las funciones externas creadas.

Y los objetivos que no se han completado han sido:

- Crear una serie de funciones dentro del DLL que permitan almacenar matrices dentro del espacio de memoria del DLL.
- Desarrollar una función externa que permita colapsar dimensiones de una matriz en menor tiempo que Vensim para matrices grandes.

6.2. Seguimiento del plan

Las predicciones del tiempo que llevaría realizar cada uno de los prototipos resultó ser no muy alejada de la realidad, obteniendo las siguientes diferencias de tiempo:

Prototipo	Tiempo previsto	Tiempo utilizado	Diferencia de tiempo
Primero	2 semanas	3 semanas	+1 semana
Segundo	3 semanas	3 semanas	+0 semanas
Tercero	3 semanas	4 semanas	+1 semana
Manuales	1 semana	2 semanas	+1 semana
Cuarto	2 semanas	4 semanas	+2 semanas
Quinto	2 semanas	1 semana	-1 semana
Total	13 semanas	17 semanas	+4 semanas

Cuadro 6.1: Comparación del tiempo previsto con el tiempo utilizado en el desarrollo.

Se tenía de tiempo para realizar el trabajo entre tres y cuatro meses, cosa que ha hecho que no se pudiera finalizar totalmente los dos últimos prototipos. Sin embargo, los objetivos principales que eran optimizar la inversión matricial y crear los manuales de instalación y de uso han sido completados exitosamente.

Esta operación es la que más tiempo requería y la que preveían que iba a necesitar cada vez más tiempo, por lo que empezaban a necesitar en GEEDS urgentemente optimizar esta operación de inversión matricial.

Sin embargo esto significa que no se ha logrado cumplir con todos los objetivos propuestos al inicio del trabajo, concretamente los relacionados con la creación del cuarto y quinto prototipo, de los cuales solo se ha logrado una compleción parcial.

Además, se han dejado asentadas unas buenas bases para poder realizar mejoras a este DLL en un futuro.

6.3. Trabajo futuro

Existen muchas posibilidades para futuros avances y optimizaciones en este proyecto. Entre las más destacables se encuentran:

- Como se ha mencionado en el Capítulo 3, en el prototipo 3 se menciona que existe la opción de llamar a una función de OpenBlas para averiguar la mejor cantidad de memoria que reservar para el vector auxiliar de datos.

  Esta operación tiene un coste de tiempo, sin embargo, en caso de que este vector auxiliar sea demasiado grande, la operación también tardará más.

  Es posible que exista una relación entre el tamaño de la matriz grande para la cual está optimizado el tamaño del vector y una matriz mucho menor que use la función a partir de la cual se ahorre tiempo al llamar a la función de averiguar el tamaño óptimo del vector antes de comenzar a operar.

  Como se explicó también, no haría falta cambiar el tamaño del vector, solo indicarle a OpenBlas que hay menos memoria reservada para el vector de la que en realidad hay.

- Crear más casos de prueba para la función `MATRIXINVERT` para poder estudiar cómo evoluciona la mejora del tiempo con matrices más grandes.

- Crear y depurar la función SUM
- Desarrollar la otra posibilidad de la función SUM para averiguar cuál funciona mejor.
- Buscar una solución a los problemas de memoria que se generan al intentar almacenar matrices y borrarlas.
- Crear una solución que permita a funciones recibir ambos tipos de SimpleMat, pero que permita a otros restringir cuál de los dos tipos recibe, de manera que no se puedan crear errores en el código debido a utilizar un tipo de SimpleMat en una operación que haga que falle.
- Implementar más operaciones en el DLL que comprendan una gran parte de las ejecuciones de los modelos creados para el proyecto LOCOMOTION.

Apéndice A

Guía de instalación

(Esta guía se presenta en la página siguiente por razones del editor de texto).

Manual for the Installation of OpenBlas and Custom Functions for Vensim

Introduction:

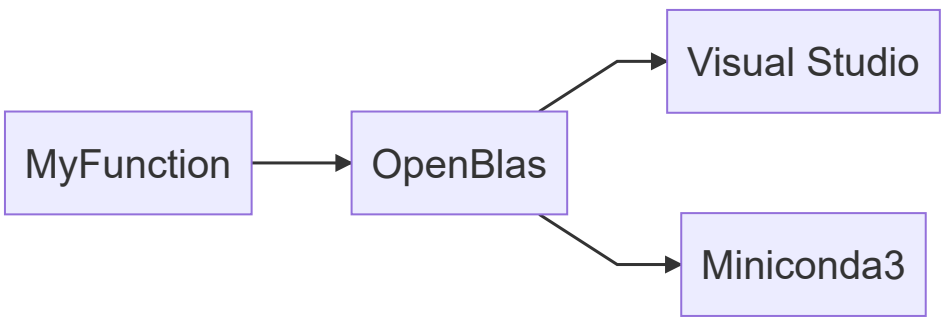
OpenBlas is a library made for optimizing the execution of mathematical operations, mainly matrix related ones. Its installation in a specific hardware configures the library so that it uses hardware specific optimizations, including parallel processing and certain CPU exclusive operations among many other minute optimizations.

This lets us execute complex mathematical operations faster than Vensim can, mainly because Vensim only uses one CPU core while OpenBlas can use more cores, as long as they are available, and use them more efficiently.

OpenBlas is used by the library that Vensim loads our custom functions from, and thus, it is necessary in order for it to function properly, among other dependencies OpenBlas requires. These dependencies are listed in this manual and they are part of the software used to install OpenBlas.

Since OpenBlas is a x64 library, this library is x64 too, so Vensim 8 or newer is required to use the custom functions.

In order to install OpenBlas, you need to install other software before, which are Miniconda 3 and any version of Microsoft Visual Studio 2015 or newer, having installed the desktop development with c++ package. This guide will cover how to install these programs too.



Miniconda 3 is a terminal that will be used to download some of the libraries OpenBlas uses and to ensure it gets installed in the correct environment.

Microsoft Visual Studio needs to be installed so that we can use some of its files for the installation. There is no need to sign in, just having it installed is enough.

The steps we will be following can be found on the official OpenBlas installation guide for windows, in the first section (`1. Native (MVSC) ABI`). However, you need to pay extra attention to certain steps in order to be able to use OpenBlas in Vensim, specifically the directories you decide to install the software in. The list of the dependencies and a command that opens a file from Visual Studio will assume that you have installed the software in the default directory.

This guide also includes the commands to install Miniconda and Visual Studio through chocolatey in case you want to do so. This is completely optional.

Miniconda 3 installation:

Miniconda can be downloaded from this link, scrolling down until you get to the section "Latest Miniconda Installer Links". The version used in the making of the library is the one for python 3.9, 64 bits. However, any version should work as long as it is 64 bits. Older version links are below the "Latest Miniconda Installer Links" section, grouped by OS.

The installation is really straightforward, follow the instructions from the installer. However, it is really important that you note the directory you install Miniconda in. You may need to run the installer as an administrator.

You can also install it through chocolatey with the command: `choco install miniconda3`

Microsoft Visual Studio installation:

Microsoft Visual Studio can be downloaded from microsoft's page. The recommended version to install is 2019, but any newer than 2015 included should work. Follow the steps of the installer. However, when prompted to select the packages to install, be sure to have "desktop development with c++" selected before continuing.

There usually are three options for downloading any version of Visual Studio: Community, Professional and Enterprise. The Community version is free and has everything we need for the installation.

In case you already had Visual Studio installed but without the package, you can just run the installer again, click modify and you will be directed to the package selection screen, where you can add new packages. Select and install the desktop development with c++ if that is the case.

You can also install the base Visual Studio with the command `choco install visualstudio2019community`, then the package with `choco install visualstudio2019-workload-nativedesktop`.

OpenBlas installation:

This next couple of steps lead to the installation itself. There will be a command that may take an hour or longer, depending on hardware. It is advised to continue installing only if you have enough time.

First get the binary packages from this link. The version used for the development of the library is 3.15, however, downloading the latest stable version available is recommended.

Downloading the latest version is as straightforward as clicking the "Download Latest Version" green button. However, to download any specific version you have to click on the folder named after the version first, then click on the "OpenBLAS x.x.xx version.zip" file to start the download.

Then unzip it and enter in the folder which was in the zip. You can know if you are in the right directory if you can find a file named “Makefile”. This directory is the one you will have to go to in the next step, when using the Miniconda command prompt.

Open the Miniconda 3 command prompt, which can be easily found by searching “Anaconda Prompt” or “miniconda3” in the windows search bar, then use ‘cd’ to navigate to the directory mentioned above, the one where OpenBlas is.

Now we need to copy and paste the following commands one by one. For an explanation on what each command does, check the OpenBlas installation guide.

```
conda update -n base conda
conda config --add channels conda-forge
conda install -y cmake flang clangdev perl libflang ninja
"c:/Program Files (x86)/Microsoft Visual
Studio/2019/Community/VC/Auxiliary/Build/vcvars64.bat"
```

To know if this command has worked properly, type the command `link` in the console. If it doesn’t return “command not found” or something similar, it has worked properly.

If this last command hasn’t worked, it may be because you don’t have Visual Studio installed.

If you don’t have Visual Studio 2019, but instead have a different year edition, change the 2019 in the command by the version of the Visual Studio you have installed.

Also, be sure to include the quotation marks when entering the command.

It may be possible to use a version prior to 2015, you can check this guide if that’s the case.

The next set of commands are the ones that need you to be in the correct directory. If you get an error related to not being in the OpenBlas directory, you need to run all commands from this point onwards again, but there is no need to run the previous ones again.

```
set "LIB=%CONDA_PREFIX%\Library\lib;%LIB%"
set "CPATH=%CONDA_PREFIX%\Library\include;%CPATH%"
mkdir build
cd build
cmake .. -G "Ninja" -DCMAKE_CXX_COMPILER=clang-cl -DCMAKE_C_COMPILER=clang-cl
-DCMAKE_Fortran_COMPILER=flang -DCMAKE_MT=mt -DBUILD_WITHOUT_LAPACK=no -
DNOFORTRAN=0 -DDYNAMIC_ARCH=ON -DCMAKE_BUILD_TYPE=Release -
DBUILD_SHARED_LIBS=ON
```

If you get `CMake Error: The source directory "...." does not appear to contain CMakeLists.txt`, you are not in the correct directory.

```
cmake --build . --config Release
mkdir ..\install
cmake --install . --prefix ..\install -v
```

Now you have a folder named install in the OpenBlas folder with the files we need to use. You can change the last two commands if you would rather install the library elsewhere, however it is portable, so you can also move the whole folder to any location you want to after installing.

This folder referred to as install in the command given will be referred to as the folder where OpenBlas is installed in the next step.

Dependencies list and location:

This is the last step for making the custom library functions work. All the dependencies must be in the same folder as the Vensim executable (by default: "C:\Program Files\Vensim"). Copy the dlls into the same folder where the vendss64.exe file is.

- C:\Users\User\miniconda3\pkgs\libflang-11.0.1-h0e60522_20210131\Library\bin\flang.dll
- C:\Users\User\miniconda3\pkgs\libflang-11.0.1-h0e60522_20210131\Library\bin\flangtri.dll
- C:\Users\User\miniconda3\pkgs\libflang-11.0.1-h0e60522_20210131\Library\bin\libomp.dll
- C:\Users\User\miniconda3\pkgs\llvm-openmp-11.0.1-h2d74725_0\Library\bin\pgmath.dll

The next file is located in a subdirectory of the folder where OpenBlas is installed.

- \install\bin\openblas.dll

If you have installed Miniconda through chocolatey, the path to the 'miniconda' folder is: C:\tools\miniconda3

Lastly, copy the DLL containing the custom functions into the folder. This file can be downloaded from https://github.com/AManteola/external-functions-vensim. The file is called 'DLLVensimExternalFunctions.dll'.

Load into Vensim:

Once the DLL is installed, open Vensim, open the 'Tools->Options' menu and in the tab 'Startup' there is an option called 'External function library (x64)'. Click that option's browse button and find the dll called 'DLLVensimExternalFunctions.dll' in the same folder as the Vensim executable is located (by default: "C:\Program Files\Vensim").

Uninstallation:

Once you have OpenBlas installed and working, you can uninstall Miniconda3 and Visual Studio. However, it is recommended that you have a backup for all the dlls mentioned in the previous step, as you won't be able to recover them without reinstalling the programs in case they accidentally get deleted by reinstalling or updating Vensim, or any other possible mishap.

How to update the custom functions library:

To update the custom functions library, you just have to swap the old dll file with the new one.

Apéndice B

Guía de uso de la biblioteca DLL

Una vez cargado el dll en Vensim, el cual se puede instalar siguiendo la guía disponible en el Apéndice A estará disponible la función `MATRIXINVERT`.

`MATRIXINVERT( matrix )` toma como argumento la matriz cuadrada que se desea invertir, y devuelve el resultado de invertir esta matriz tal y como lo haría la función `INVERT MATRIX( matrix , n )` de Vensim, con la diferencia de que ofrece un rendimiento mucho mejor para matrices de gran tamaño.

Apéndice C

Guía de instalación y uso de la plataforma de desarrollo

La versión de Code::Blocks usada en el desarrollo de este trabajo ha sido la versión 20.03mingw. Es importante que sea la versión que incluye MinGW dado que es el compilador de C usado. [15]

Para poder compilar el código se necesita tener OpenBlas instalado como se puede hacer siguiendo la guía del Apéndice A.

Una vez se tiene CodeBlocks descargado y el proyecto, se ha de abrir desde CodeBlocks el archivo de extensión `.cbp` que se puede encontrar en la carpeta raíz del proyecto. Esta extensión significa «code blocks proyect».

Una vez cargado el proyecto, click derecho sobre él en el workspace y abrir las *Build Options*. Ir a la pestaña de *Search directories* y añadir en la pestaña de *Compiler* las siguientes carpetas:

- %Ubicación de OpenBlas instalado %\include\openblas
- %Ubicación de Miniconda3 %\pkgs\libflang-11.0.1-h0e60522_20210131\Library\bin
- %Ubicación de Miniconda3 %\Library\bin

En la pestaña de *Linker* hay que añadir los mismos pero añadiendo extra los siguientes directorios:

- %Ubicación de OpenBlas instalado %\bin
- %Ubicación de OpenBlas instalado %\lib

Es posible que el directorio:

%Ubicación de Miniconda3 %\pkgs\libflang-11.0.1-h0e60522_20210131\Library\bin

no tenga exactamente ese nombre. Sin embargo se puede encontrar al realizar la búsqueda de flang.dll desde el directorio en el que se encuentra Miniconda3.

Cuando se ha compilado el proyecto, se puede encontrar el dll compilado en la carpeta:

%Ubicación del proyecto %\bin\Debug

Apéndice D

Repositorios de documentación y software

El código fuente junto al manual y el dll compilado se pueden encontrar en:

`https://github.com/AManteola/funciones-externas-vensim`

Apéndice E

Licencia de uso de OpenBlas

Bibliografía

[1] *Performance Comparison of OpenBLAS* and Intel® Math Kernel Library*,
SOFTWARE.INTEL.COM.
Recuperado 15/09/2021,
de https://software.intel.com/content/www/us/en/develop/articles/performance-comparison-of-openblas-and-intel-math-kernel-library-in-r.html

[2] *Vensim Software*,
VENSIM.COM.
Recuperado 15/09/2021,
de https://vensim.com/vensim-software/

[3] *Basic Linear Algebra Subprograms*,
WIKIPEDIA.ORG.
Recuperado 15/09/2021,
de https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

[4] *Linear Algebra Package*,
WIKIPEDIA.ORG.
Recuperado 15/09/2021,
de https://en.wikipedia.org/wiki/LAPACK

[5] *Intel oneAPI Math Kernel Library*,
INTEL.COM.
Recuperado 15/09/2021,
de https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html#gs.binq1p

[6] *Comparison of Linear Algebra Libraries*,
WIKIPEDIA.ORG.
Recuperado 15/09/2021,
de https://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries

[7] *Venisim manual for external functions*,
VENSIM.COM.
Recuperado 15/09/2021,
de https://www.vensim.com/documentation/dss_external_fn.html

[8] *Vensim Online Documentation*,
VENSIM.COM.
Recuperado 15/09/2021,
de https://www.vensim.com/documentation/index.html

[9] *OpenBlas installation for Windows*,
GITHUB.COM/XIANYI.
Recuperado 15/09/2021,
de https://github.com/xianyi/OpenBLAS/wiki/How-to-use-OpenBLAS-in-Microsoft-Visual

[10] *The LAPACKE C Interface to LAPACK*,
NETLIB.ORG.
Recuperado 15/09/2021,
de https://www.netlib.org/lapack/lapacke.html

[11] *GEEDS homepage*,
GEEDS.ES.
Recuperado 15/09/2021,
de https://geeds.es

[12] *LOCOMOTION homepage*,
LOCOMOTION-H2020.EU.
Recuperado 15/09/2021,
de https://www.locomotion-h2020.eu

[13] *MEDEAS homepage*,
MEDEAS.EU.
Recuperado 15/09/2021,
de https://www.medeas.eu

[14] *Visual Studio*,
MICROSOFT.COM.
Recuperado 15/09/2021,
de https://visualstudio.microsoft.com

[15] *Code::Blocks*,
CODEBLOCKS.ORG.
Recuperado 15/09/2021,
de https://www.codeblocks.org/downloads/binaries/

[16] *Miniconda3*,
CONDA.IO.
Recuperado 15/09/2021,
de https://docs.conda.io/en/latest/miniconda.html

[17] *MinGW*,
MINGW-64.
Recuperado 15/09/2021,
de https://www.mingw-w64.org